Head First iPhone Development

深入浅出iPhone开发



精通
Objectiv-C、
Interface Builder
和Xcode



设计热销
的App



玩转iPhone的
GPS和照相机



编织出一道
美味的多视
图调酒应用



看看迈克如何用
一个iPhone Twitter
App拯救他的爱情
生活

[美] Dan Pilone & Tracey Pilone 著
鲁成东 译

O'REILLY®
人民邮电出版社
POSTS & TELECOM PRESS

深入浅出iPhone开发

Mobile Development/Apple

你会从本书中学到什么？

假设你有一个绝妙的创意，可以用来做一个超酷的iPhone应用，那么该从何处下手呢？《深入浅出iPhone开发》这本书就可以帮助你，让你不费吹灰之力就能搭建起应用程序。你将迅速地学会使用iPhone SDK工具，其中包括Interface Builder和Xocde，并能掌握Objective-C的编程原则，从而让你的应用更为出众。本书中还包括了完整的学习体验，以帮助读者创作出引人注目、销售火爆的iPhone应用程序。



为什么这本书看起来如此与众不同？

我们深知大家的时间宝贵，不应纠结于那些新概念之上。因此，我们利用认知科学与学习理论的最新研究成果，在本书中打造了一种可以进行全方位感知的新型学习体验，其中大量运用了丰富的视觉效果，并根据人脑的工作方式对它们进行精心安排，而完全取代了那些让你昏昏欲睡的连篇累牍的大段文字。

"关于这本书最棒的一点，就是它那种简单却又循序渐进的讲解方式。它并不会试图去教会你所有的东西——而只是用一种友好的、对话式的方法让你学会正确地构建iPhone应用程序。对于那些已经了解如何写代码、仅仅想要直接感受到iPhone开发乐趣的人来说，这是一本及时雨式的好书。"

——Eric Shephard，Syndicomm拥有人

"《深入浅出iPhone开发》的作者为你清晰地打造出此书，令你不必预先对Macintosh开发工具有所了解，就可以轻松地创建、使用以及学习iPhone的技术。"

——Joe Heck，Seattle Xcoders创始人

"本书可谓是一本授人以渔的好书！在我们当中颇有一些人吃够了学习iPhone开发的'苦头'之后，现在看到别人苦尽甘来，不禁让我们更加自觉命苦。"

——Mike Morrison，Stalefish Labs创始人

O'REILLY®
oreilly.com
headfirstlabs.com

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

分类建议：计算机/移动开发
人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-24470-3
9 787115 244703 >
ISBN 978-7-115-24470-3
定价：89.00 元

深入浅出iPhone开发



[美] Dan Pilone & Tracey Pilone 著
鲁成东 译

O'REILLY®

Beijing • Cambridge • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc.授权人民邮电出版社出版

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

深入浅出iPhone开发 / （美）丹皮洛内（Pilone,D.）
，（美）皮洛内（Pilone,T.）著；鲁成东译．-- 北京：
人民邮电出版社，2011.5
ISBN 978-7-115-24470-3

Ⅰ．①深… Ⅱ．①丹… ②皮… ③鲁… Ⅲ．①移动通
信－携带电话机－应用程序－程序设计 Ⅳ．
①TN929.53

中国版本图书馆CIP数据核字(2010)第238384号

版 权 声 明

©2008 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Posts & Telecom Press, 2010. Authorized translation of the English edition, 2010 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc.出版 2008。

简体中文版由人民邮电出版社出版，2010。英文原版的翻译得到 O'Reilly Media, Inc.的授权。此简体中文版的出版和销售得到出版权和销售权的所有者 —— O'Reilly Media, Inc.的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

深入浅出 iPhone 开发

◆ 著　[美]Dan Pilone　Tracey Pilone
译　鲁成东
责任编辑　汪　振

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：787×1000　1/12
印张：45
字数：540 千字　2011 年 5 月第 1 版
印数：1 - 3 500 册　2011 年 5 月北京第 1 次印刷
著作权合同登记号　图字：01-2009-7676 号

ISBN 978-7-115-24470-3

定价：89.00 元

读者服务热线：(010)67132705　印装质量热线：(010)67129223
反盗版热线：(010)67171154

内容提要

想要学会如何调制鸡尾酒吗？或者试一试追捕逃犯、主持正义？至少你应该会想要追上自己心仪的女生吧？无论你对于以上几种的哪一项感兴趣，都一定要看看这本书。这些内容本书全都涉及到了，而且在此之外，你还能学会iPhone应用开发呢！

本书是针对iPhone开发的初学者设计的，以几个应用实例的开发为例，循序渐进地对iPhone开发的各个方面进行了讲解。由于内容是根据实际程序中所使用到的技术进行编排的，同时又以各种不同方式展示内容，因此能令读者快速上手，又不失深刻记忆。

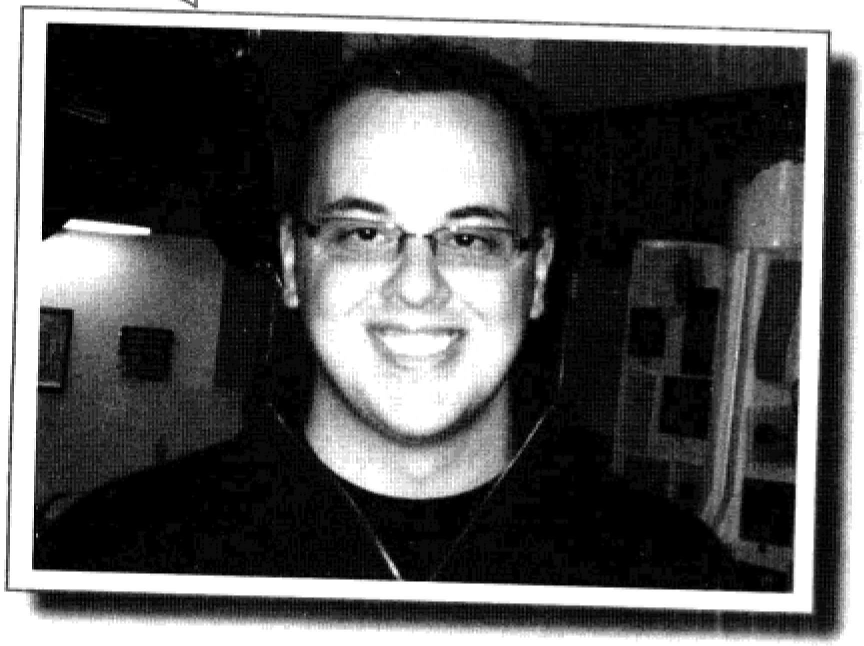
书中风格与其他Head First系列一脉相承，一定能让读者轻松学会iPhone开发。

献给我最好的朋友暨丈夫——Dan，以及Vinny与Nick这对天下最好的孩子。

— Tracey

谨以此书献给我的全家：给爸爸妈妈，没有你们就没有这一切；给总是和我抬杠的兄弟，以及我的妻子和儿子们，你们不仅容忍了这一切，还帮助我来完成这本书。

— Dan





Dan Pilone是一名软件架构师，供职于Vangent公司，曾经领导过海军研究实验室、UPS、Hughes和NASA的一些软件开发团队。他曾在华盛顿特区的国家天主教大学为本科生和研究生教授软件工程课程。

Dan之前也写过Head First系列书籍，分别是《Head First软件开发》和《Head First代数》，所以他能够适应这种略微有异寻常的风格，但这还是第一本涉及到代理（或经纪人）的书。更吓人的是，Tracey为了完成这本书，变成了一个夜猫子以及苹果的女粉丝。

Dan在弗吉尼亚理工获得了计算机科学学位以及数学第二学位，他还是O'Reilly的iPhone开发培训班的讲师之一。

Tracey Pilone首先要感谢她的合作者和丈夫，感谢他分享本书，并意志坚定地迟迟不睡以确保工作正确完成。

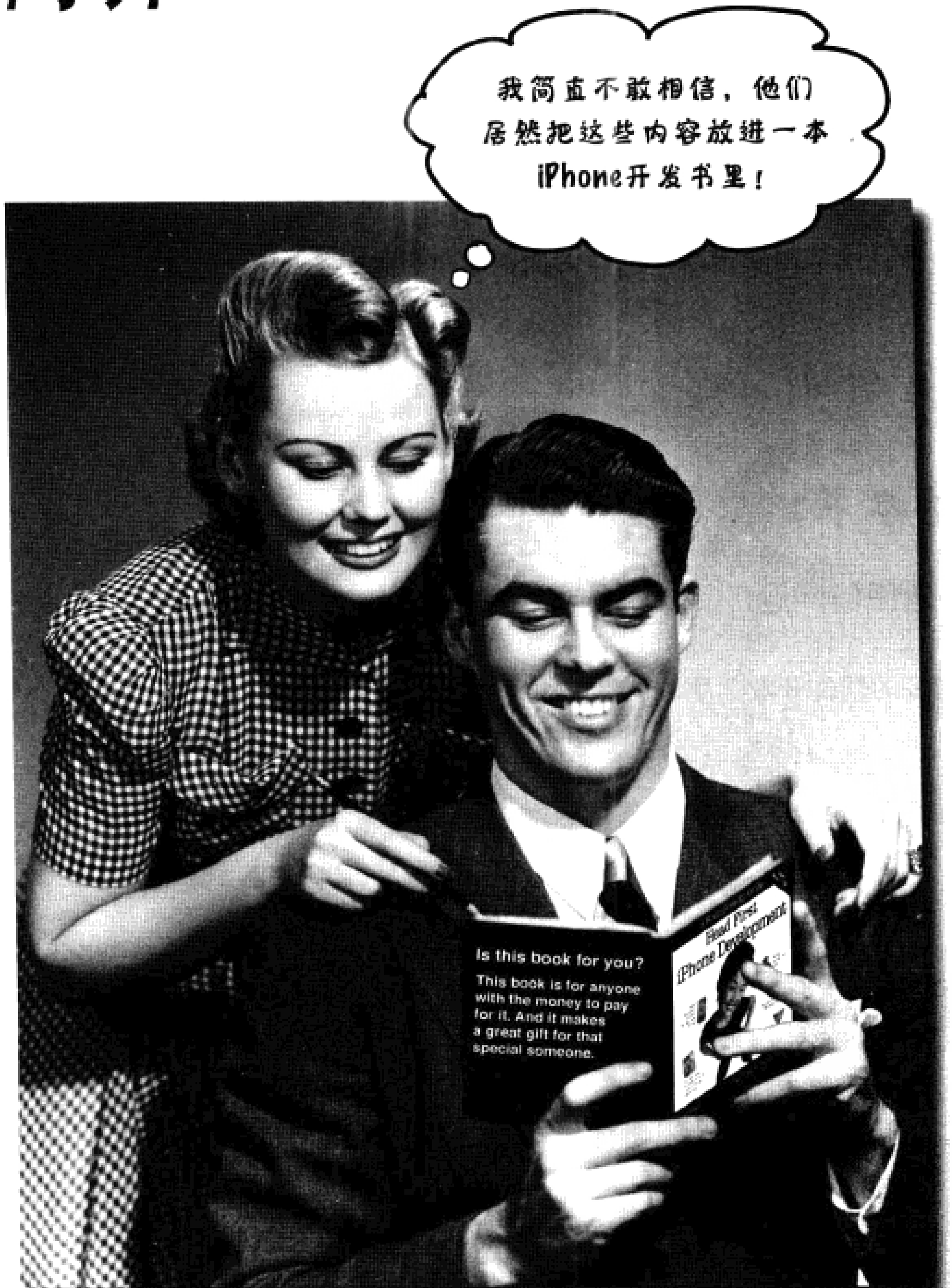
她是一个自由职业技术作家，为海军的任务规划与射频分析做支持，也是《Head First代数》的作者。

成为作家之前，她在华盛顿特区周边的大型商业建筑工地做过几年建筑经理。那时她所在的团队负责协调办公室建筑的设计与建造，所用到的工程与管理技巧在某种程度上有助于写作Head First书籍。

她从弗吉尼亚理工获得了土木工程学位，在弗吉尼亚大学获得教育学硕士。

如何使用这本书

简介



在本节中，我们回答了这个惹人注意的问题："到底为什么要将这些内容放进一本iPhone开发书籍里？"

哪些人该读这本书？

如果你对下列问题的回答都是“是”的话：

1. 你以前有开发经验吗？

如果你有一些面向对象的经验，肯定会有帮助。Mac开发经验也会有所帮助，但绝不是必需的。

2. 你想要学会、理解、牢记并能够应用重要的iPhone设计与开发理念，从而编写自己的iPhone应用程序，并在App Store中出售它们吗？

3. 与干巴巴又枯燥的学术讲演比起来，你更喜欢令人兴奋的餐桌对话吗？

那这本书就是为你准备的。

哪些人最好远离这本书？

如果你对下列任何问题的回答是“是”的话：

1. 你是个彻底的软件开发新手吗？

看看介绍面向对象开发的书，看完了之后再来加入我们iPhone村。

2. 你已经在开发iPhone应用了，只是想找一本Objective-C的参考书吗？

3. 你害怕尝试不同的事物吗？你是不是宁愿去拔牙也想穿得与众不同？你是否会认为一本技术书籍里如果有代理掺和进来就不可能是本正经书呢？

那么，这本书不适合你。

[市场部门提示：这本书适合任何人，只要你有信用卡，或者有现金也行。 - Ed]



我们知道你在想什么。

“这会是一本正经的iPhone开发书籍吗？”

“弄这些画到底是为什么啊？”

“像这样我能学到东西吗？”

而且我们也知道你的大脑在想什么。



你的大脑渴望新奇的东西。它总是在搜寻、等待一些不同寻常的东西。这就是它的工作方式，正是如此才能帮助你存活下来。

那么，大脑是如何处理你所遇到的日常、平凡而一般的事物的呢？随便怎么处理，只要不影响到它的真正工作——记录那些重要的事——就可以。它才不会费心去储存那些枯燥的东西；它们从来不会越过“那显然并不重要”的过滤防线。

那大脑是如何知道一件事重不重要的呢？假定你去徒步旅行，而一只老虎突然出现在你面前。在你的头脑和身体里会发生什么呢？

神经元火烧火燎；情绪万分激动；化学物质汹涌澎湃。

这时你的大脑就知道了……

这一点非常重要！千万别忘了！

但是假设你在家里或者图书馆。这里安全、温暖，也没有老虎。你在学习中，为考试做准备；或者试着学习一些艰深的技术内容，而你的老板认为只要一个礼拜或最多10天就能学会。

这就有问题了。你的大脑试着给你帮忙。它试图确保这些明显不重要的内容不会浪费掉稀缺的资源——那些最好用在存储真正重要的事物上的资源。比如老虎、火焰的危险，或者是为什么你绝不应该再穿短裤去滑雪。

没有一种简单的方法可以告诉你的大脑：“嗨，脑兄，谢了。不过不管这书多闷，也别管我现在的情绪指数有多低，我真的需要让你把这玩意儿记住。”



我们认为“Head First”（深入浅出）系列图书的读者是那些初学者

那么，学习需要些什么呢？首先，你必须要理解学习的内容，然后是确保你不会忘了它。学习并不是要把一堆事情塞到你的脑袋里去。认知科学、神经生理学以及教育心理学的最新研究成果表明，学习所需的内容远远要比书本上的文字内容多。我们知道什么会令你的大脑运转起来。

一些Head First学习法则：



让学习内容可视化：图像远比文字本身更容易记住，会令学习更有效率（在回忆与迁移方式的学习中高达89%），它还会令事物更容易理解。

将词语放在与其相关的图形之中或者旁边：与放在底部或者另一页上相比，这样能使得学习者在解决与内容相关的问题时，成功的机会增至两倍之多。



使用个性化的对话风格：在最近的研究中，如果将学习内容用第一人称的对话方式直接与读者交谈，要比那些用了正式语调谈话的学生，在之后的考试中成绩好40%。讲故事，而不要说教。使用通俗的语言，不要太一本正经。想一想，你在晚宴上是对撩人的舞伴感兴趣，还是更关注演讲？

让学习者更深入地思考：换句话说，除非你主动拨动你的神经元之弦，在你的脑袋里就不会发生任何事。读者必须是主动的、专心的、好奇的，有创意地去解决问题、得出结论，并产生出新的知识。而为了达到这一点，你需要挑战、练习以及发人深省的提问，还有同时涉及大脑两半球和多种感官的活动。

吸引读者的注意力并保持住：我们都有过一种经历，那就是“真的想学习，但是一打开书就犯困”。你的大脑只会对超常的、有趣的、奇怪的、吸引眼球的、出乎意料的事情给予关注。学习一门新的、艰深的技术并不一定就必须是枯燥乏味的。如果能做到这一点的话，你的大脑的学习速度就会快得多。



触动他们的情绪：现在我们知道，你记忆事物的能力很大程度上依赖于它的情绪内容。你会记住你在乎的东西。你对某些事情有感觉时就会记得牢。不，我们并不是在谈论一个关于男孩和他的狗的催人泪下的故事。我们谈论的是惊讶、好奇、有趣还有“这到底怎么回事？”之类的情绪，以及当你解决一个谜题时随之而来的那种“我能行！”的感觉，学会其他人认为很难的一件事，或者是意识到你知道一些事情，而那个念工程的鲍勃还不知道，会让你觉得“我技术比他还强”。

元认知：关于思考的思考

如果你真的想学习，而且想更快速、更深入地学习，就要注意你是如何集中注意力的。思考你的思考方式。学习你的学习方法。



我们中大多数人在成长过程中，都没有上过关于元认知或者学习理论的课程。大人们都期望我们学习，但很少有人教会我们怎么学习。

但你现在拿着这本书，我们认为你是真的想要学习iPhone开发技术，而你很可能并不想花很多时间。由于将来你会构建许多应用，所以需要记住你读过的东西。因此，你必须理解它们。要最大限度地利用这本书、其他的书或学习经验，就要对你的大脑负责。关注当前的内容。

要做到这一点，关键技巧就是让你的大脑将正在学习的材料当作真正重要的东西，对你性命攸关。就像一只老虎一样重要。否则，你就会与你的大脑陷入一场旷日持久的战争，竭力让它记住这些新内容。

那么，到底怎样才能让你的大脑把iPhone开发当作一只饥饿的猛虎呢？

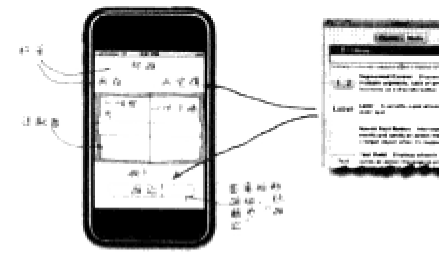
有一种缓慢而又沉闷的方法，也有一种更快、更高效的方式。慢的方法就是单调地重复。你肯定知道，只要一遍又一遍地往脑子里塞，就是最无聊的东西你也能学会并记住。经过足够的重复，你的大脑就会认为："这东西看起来对他并不重要，可是他一遍一遍又一遍地重复看它，估计还是挺重要的。"

更快速的方法是做一些能增加大脑活跃程度的事，特别是不同类型的大脑活动。前一页所讲的就是答案的大部分内容，它们都业已证明可以帮助大脑按你的需要工作。例如，研究表明把词语放在它们所描述的图画中（相反的做法是放在页面中其他部分，比如标题或者正文中），会导致大脑试图找出词语和图画之间的关联，从而导致更多的神经元兴奋起来。而更多的神经元兴奋 = 更有机会让你的大脑认为这是值得注意的事情，从而可能将其记录下来。

对话的方式对此也有帮助，因为人们认为自己在进行对话时，会给予更多的注意力，这是由于他们想跟上节奏并且说出自己那部分对话。令人惊奇的是，你的大脑并不在乎"对话"的另一方是一本书！相反地，如果行文风格是正式而又干巴巴的，你的大脑就会把它当成是坐在满是被动的观众的屋子里听演讲。那就没必要保持清醒了。

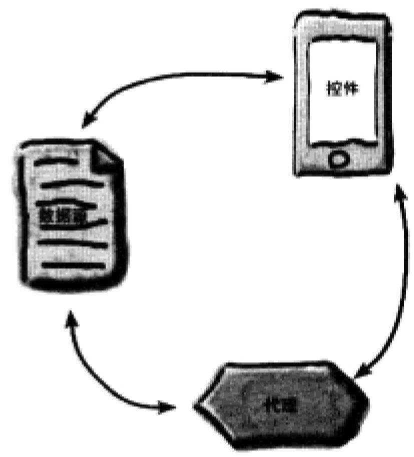
不过，图片和对话风格才只不过是个开头而已。

这就是我们所做的：



我们使用了图画，因为大脑更适应形象刺激而不是文字。仅仅从大脑的角度来说，一幅图画真的顶得上一千句话。在需要让文字与图画同时起作用时，我们会把文字嵌到图画中，因为文字出现在相关的事物当中，与放在标题或者正文中的什么地方相比，会令大脑的工作效率更高。

我们会进行复述，将同样的东西用不同的方式、不同类型的媒介针对不同的感官反复讲述，来增加内容被组织起来、从而进入你大脑的多处区域的机会。



我们会以出乎意料的方式使用概念和图画，因为大脑更期待新奇事物，我们还会令图画与创意更有情绪色彩。这样可以导致你对某些事有感觉，从而更可能被记住，即使这些感觉可能只不过是一点点幽默、惊讶或者有趣。

我们会使用一种个性化的对话风格，因为大脑认为在进行对话时会给予的关注，远超过被动地听讲座。即使你是在阅读，你的大脑仍然会这样做。

我们会引入大量的活动，因为大脑在你做事的时候，能比阅读时更好地学习和记忆。我们设计的练习，会让大多数人都感到难易适中。

我们使用了多种学习方式，因为你可能喜欢按图索骥的过程，而其他人想要首先理解整体图景，另一些人只是想看到一个例子。但不管你喜欢哪种学习方式，每个人都会从多种角度去领略相同的内容，并从中受益。

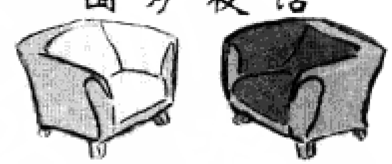
我们为大脑两个半球都准备了内容，因为使用大脑越多，就越容易学习和记忆，也能更长时间地集中注意力。使用大脑的一侧常常意味着让另一半有机会休息一下，所以你就可以在学习中更长时间地保持学习效率。

关键
要点



我们还加入了体现多角度观点的故事和练习，因为大脑在被迫进行评价与判断时，会学习得更加深入。

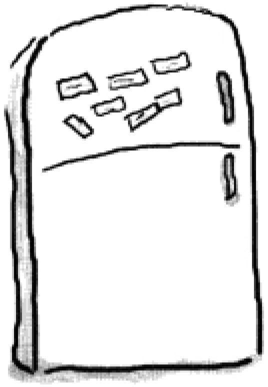
围炉夜话



我们通过各种方式来引入挑战性，比如通过习题，或者通过问一些答案并不是显而易见的问题等，因为大脑在必须解决问题时会更好地学习和记忆。想想就知道了——你是不会看看别人在健身房锻炼就能变得苗条的。不过我们会竭力保证，当你努力工作时，力气是花在了正地方。你不会浪费一个多余的神经元，去处理一个难以理解的例子，或者去破译晦涩难懂、充满术语甚至过分简洁的文字。

我们在故事、例子、图画等当中使用了人物。因为，你的大脑会对人给予更多关注，超过对物的关注。





这样做就能让你的脑子转过弯来

行了，该我们做的部分已经做完了。现在该看你的了，下面这些技巧只是一个起点；倾听大脑的反应，来找出哪些技巧对你起作用，哪些没用。尝试些新鲜事物。

把这些剪下来，贴到你的冰箱上。

① 放慢速度。你理解的越多，需要记忆的就越少。

不要仅仅去阅读。要不时停下来思考。当书里提出问题时，不要马上看答案。要当成真的有人问你这个问题。你让大脑思考得越深入，你学习和记忆的效果就越好。

② 做练习。自己写笔记。

题目我们都放进书里了，但是如果我们替你把练习做了，那就好像是让别人替你健身一样。所以不要只是盯着这些习题。拿起笔来。有大量的证据表明，学习的同时进行一些身体活动可以促进学习效果。

③ 阅读“有问必答”部分。

这部分是很重要的，并不是可有可无的装饰栏——它们是核心内容的一部分！不要跳过这些内容。

④ 把本书当成睡前的最后读物，或者至少是最后一本有难度的书。

学习过程的一部分（特别是长期记忆的转移），发生在你放下书本之后。大脑需要有自己的时间，来进一步处理信息。如果你在处理过程中又加入了新的内容，你刚学过的东西会丢掉一部分。

⑤ 饮水。大量饮水。

大脑在液体充分浸泡下工作效率最高。脱水（这甚至会在你感到口渴之前发生）会降低认知能力。

⑥ 说出来，大声点。

说话会激活大脑的另一个区域。如果你试图理解某些事物，或者想增大以后记住它的机会，可以把它大声说出来。更好的方法，是把它向其他人大声解释一遍。你会学得更快，可能还会发现一些在阅读时没有注意到的地方。

⑦ 倾听大脑的反应。

时刻注意你的大脑是否超载。如果你发现自己的理解开始变得蜻蜓点水一般肤浅，或者刚看过的东西就忘记了，那就该休息了。一旦你越过某个点，那把更多东西塞进脑子里也不会学得更快了，更有甚者可能会破坏学习的过程。

⑧ 用心去感受事物！

大脑需要知道哪些事是重要的。沉浸到故事中去。自己为照片起个标题。哪怕是为一个拙劣的笑话哭笑不得，都比漠然无觉要强得多。

⑨ 去创造事物！

把这些应用到你的日常工作中；在你的项目中做决定时，使用你学到的知识。你一定要做点什么，好在书本习题和动手操作的范围之外获得经验。你需要的，只是一支铅笔和一个待解决的问题……这个问题有可能会从你为了应付考试而学习的工具和技术中获益。

必读

这是一本进行学习体验的书，而不是参考大全。我们特意去掉了一些东西，以免它们妨碍书中正在学习的内容。第一次翻开这本书，你得从头开始读，因为书中对于你已经看到和学会了哪些内容，都是有假设的。

我们会在刚开始的第1章就构建一个应用。

你能相信吗，即使你以前没有给iPhone开发过任何程序，也可以直接开始构建应用。你还会学到如何使用iPhone的开发工具。

到本书结束之前，我们都不会去管如何把你的应用提交到App Store上。

在这本书里，你可以很好地学到如何创建iPhone应用，而不用去操心那些为了把你的应用程序发送出门而必须要做的打包、分发之类的事。不过我们也知道，大家操心费力地学这个iPhone应用开发到底是为了什么，所以，我们在末尾的一个附录里介绍了整个过程（还有所有容易犯错的地方）。

我们会始终保持内容都可以在模拟器上构建与测试。

iPhone SDK自带一个超级棒（而且是免费的）的工具，可以在你的电脑上直接测试iPhone应用。这就是模拟器。模拟器令你能够直接实验你的代码，而不必操心进入App Store或者得买个真实设备什么的。不过，它也有一些局限性。iPhone上有一些很酷的功能，比如加速计和指南针，在模拟器上无法使用。所以在本书中我们不会去太详细地介绍这些内容，因为我们要确保你可以快速而轻松的创建并测试应用。

动手操作可不是选答题。

练习和动手操作内容不是添头；它们是本书核心内容的一部分。其中有一些会帮助记忆，有一些帮助理解，还有一些有助于你将所学内容活学活用。不要跳过这些练习。哪怕是填字游戏都很重要——它们会帮助你把概念收入脑海中，就像你在PMP考试中见到的那样。但更重要的是，它们是一个很好的机会，令你的大脑以一种不同的方式思索刚学到的单词和术语。

重复是刻意的，也是重要的。

Head First系列与其他书籍相比有一个显著的区别，就是我们想让你真的学会它。所以我们希望你在读完这本书后，能记住你学过的东西。大多数参考书不会把保持记忆作为目的，但本书是一本学习用的书，所以你将会见到一些相同的概念不止一次地出现。

脑力练习是没有答案的。

有一些脑力练习是没有标准答案的；而对另外一些来说，让你自己来判断答案是否正确、何时正确本来就是脑力活动中的一部分。在脑力练习中，有一些会有提示，为你指出正确的方向。

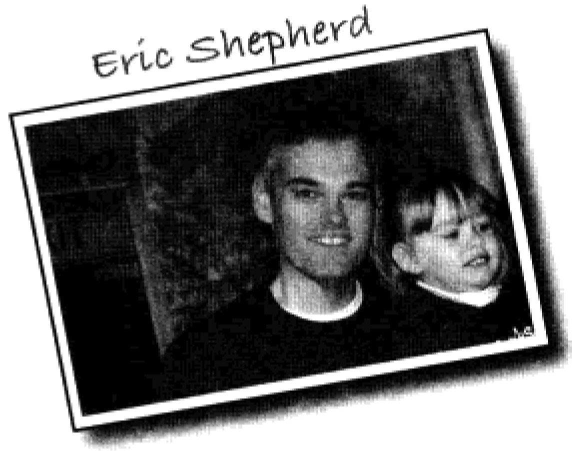
系统需求

要为iPhone进行开发，需要一台基于Intel CPU的Mac电脑，仅此而已。我们写作本书，使用的是Snow Leopard（10.6）和Xcode 3.2。如果你用的是Leopard（10.5），Xcode的版本也更老，有一些地方就会引发错误，我们会尽量把这些问题指出来。对于一些高级功能，比如加速计和照相机等，你得有一个真实的iPhone或者iPod Touch才能使用，而且要注册成为开发者。在第1章中，我们会指点你获得SDK和苹果公司的文档，所以现在不要为此担心。

技术审核团队







技术审核者

我们这本书有一个令人惊叹的精英技术审核组。他们的工作完成得非比寻常，我们对他们的杰出贡献表示万分感谢。

Joe Heck是一名软件开发者、技术经理、作家以及培训师，从事计算机相关工作达25年，自从iPhone开发平台的首次beta发布以来，一直为其进行开发。Joe受雇于Walt Disney互动媒体集团，从事各种技术和开发平台上的工作，并协助开发了Disney的iPhone游戏“Fairies Fly”。他是西雅图Xcoder开发者组织的创始人，这个组织的工作是为西雅图地区的Macintosh和iPhone开发提供支持，他还是SeattleBus这个iPhone应用的作者，这是一个提供西雅图公交运输的实时到站、发车时刻的应用（在iPhone App Store中可以找到）。他对iPhone知之甚详，帮助我们确保本书在每个技术细节上都无懈可击。他对细节的关注意味着我们的每个嘻嘻哈哈的答案都是完整无误的。

Eric Shepherd从9岁起就开始编程，从未停休。他从1997年开始成为一名技术作者，编写开发者文档，目前是Mozilla的开发者文档组长。在业余时间里，他为老式电脑Apple II编写软件——因为他觉得自己的日常工作不够酷——还有就是花时间陪他的女儿。Eric的审核意见对我们帮助极大。他令我们的任何笔误以及代码中出现的错误都被发现并改正。他审核极其详尽,实际上他把代码全都运行起来了，这使得其他人都不会再碰到他处理过的问题。

Michael Morrison是一名作家、开发者，他的著作包括*Head First JavaScript*、*Head First PHP & MySQL*。Michael是Stalefish实验室（*http://www.stalefishlabs.com*）的创始人，这是一个寓教于乐的公司，专注于游戏、玩具和互动媒体，其产品中就包括了几个iPhone应用。Michael很多时候都戴着头盔，是为了便于玩滑板、打冰球或者调试iPhone程序。由于他同时具有iPhone和Head First的经验，所以在两方面对我们都帮助匪浅。无论给出哪个方面的评审意见，他的态度都非常友善，在发现排版问题的时候还会给出一个解决方案，这令得按照这些意见进行修改变得十分简单。

他们三个全都是在快要结束的时候才加入进来的，在短短的时间中完成了海量的评审工作，我们对此表示非常感谢。非常感谢！

致谢

我们的编辑：

感谢Courtney Nash，是你从一开始就支持帮助我们，一直到最终成果出现，通常这需要很久，不过我们可没用太长时间！她始终敦促我们，确保这本书成为一本Head First书籍，虽然有些时候并不那么容易做到，但她始终如一。她知道我们说的是哪一章。



Courtney Nash



Brett McLaughlin

同样要感谢Brett McLaughlin，是你在回复一条网络聊天信息的时候说到“你们觉得Head First iPhone怎么样？”，才使得我们开始写作此书，并最终完成了它。他还在整本书的写作过程中扮演了学习者的角色(顺利完成，偶尔有点抱怨)，对开始数章的起步工作给了很大的帮助。



Mark Reese

致O'Reilly的团队：

感谢Karen Shaner掌控了整个技术审核过程，就是在最后的时候有点速度过快。也向Laurie Petrycki致谢，感谢你信任我们，让我们在刚刚完成一本Head First书籍之后不到一年就开始写另一本。最后，要谢谢我们的设计编辑Mark Reese在图形和排版方面给我们的帮助。

致我们的家人和朋友：

感谢Pilone家和Chadwicks家的所有人，一直到10月之前，在我们忙于这本书的时候，你们忍受了许多急躁情绪并给予我们所需的支持，让我们能够成长起来写完这本东西。感谢Dan的兄弟Paul，是你不停地念那些“真的，Mac特棒”之类的咒语，让Dan有信心去买了一台，从而明白OS X开发到底是怎么回事。

感谢Vinny和Nick，在过去几个月中，你们颇承受了一些“颠沛流离”的生活，不过现在好了，他们马上就要夺回爸爸妈妈的时间了。他们都想要iPhone。

感谢我们的朋友们，你们听了许多关于要搞定这本书的唠叨，并时不时地帮我们带上几个小时的孩子，所以我们才能完成此书，你们也在需要时给予了我们许多鼓励！

最后，虽然听起来很傻，还是要感谢苹果公司，因为如果没有iPhone这么独一无二、能改变游戏规则的设备，哪里会有这本书！

Safari®在线书库

Safari® 在线书库是一个点播式的数字图书馆，让你可以轻易地在超过7500部技术图书和视频中进行搜索，迅速找到你需要的答案。

订阅之后，你就可以在我们的图书馆中任意地在线阅读书籍与观看视频。你可以在手机和移动设备上阅读书籍，还可以在新书印刷出版之前就抢先阅读，甚至可以得到独家权限去查看正在写作中的手稿并向作者进行反馈。可以复制、粘贴示例代码，编排你的收藏夹，下载书籍章节，对关键段落标书签，创建笔记，打印书页，等等。大量功能会为你节省时间、让你从中受益。

O'Reilly媒体公司已经将此书英文版上传到Safari在线书库中。要想得到对此书以及类似主题书籍的完全数字访问权限，无论它们是由O'Reilly还是其他出版社出版的，都可以免费在下面的网址注册：*http://my.safaribooksonline.com*。

目录（概览）

	简介	xxi
1	新手上路：开始移动生活	1
2	iPhone应用的模式：你好，@twitter	37
3	面向iPhone的Objective-C教程：Twitter需要多姿多彩	89
4	多视图：表格中的视图	131
5	Plist与模式视图：对应用千锤百炼	185
6	数据的保存、编辑与排序：人人都能当编辑……	239
7	标签栏与Core Data：企业应用	303
8	用Core Data迁移并优化数据：一切都改变	377
9	照相机、地图开发包和Core Location：现实世界中的证据	431
i	孑余：首要的6件事（我们未涉及的）	487
ii	发布应用所需的预备工作：准备好进入App Store	503

详细目录

简介

关于iPhone开发的思考。你准备学点东西，可是你的脑子总是捣乱、开小差。你心里想："应该腾出脑子想点更重要的事，比如去滑雪怎么样。"那么你该如何让自己从心里相信，学会开发自己的iPhone应用有那么重要呢？

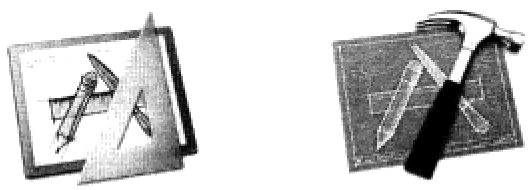
哪些人该读这本书？	xxii
我们知道你在想什么	xxiii
元认知：关于思考的思考	xxv
这样做就能让你的脑子转过弯来	xxvii
必读	xxviii
技术审核团队	xxx
致谢	xxxi

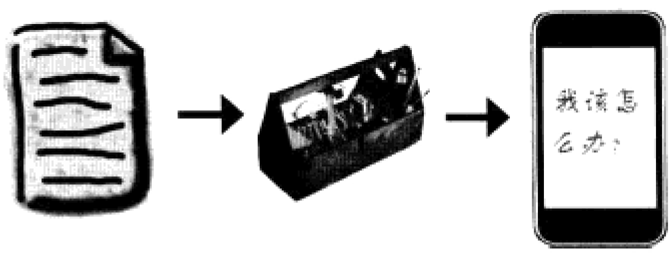
【拼吾爱-最新编程资源的分享下载站】

1 新手上路

开始移动生活

iPhone改变了一切。 它是游戏平台，又是个人信息管理器，还是一个全功能网络浏览工具，哦，对了，还是电话。iPhone是很久以来，出现的最令人激动的设备之一，而且随着App Store的开放，它也成了独立开发者的一个机会，令他们可以在世界范围内和那些大牌软件公司同场竞技。而发布你自己的应用程序所需的一切，仅仅是几个软件工具、一定的知识以及满腔热情。苹果公司提供了软件，而本书将带给你所需的知识。我们相信你一定准备好了满腔热情。









App Store甚嚣尘上，其中充斥着大量的财富…… 2
移动应用程序可不仅仅是移植桌面应用那么简单 3
iPhone应用剖析 5
迈克有点左右为难 6
第一眼留个好印象 7
一切都从iPhone SDK开始 8
Xcode包含有应用模板，可以帮你入门 10
Xcode是iPhone项目的轴心... 12
...并在编写应用的过程中无处不在 13
构筑界面，靠的是……Interface Builder 14
向视图中添加按钮 16
iPhone模拟器让你可以在Mac上测试应用 17
怎么回事? 22
用Interface Builder将UI控件与代码关联起来 23
Interface Builder会列出组件可以触发哪些事件 24
UI元素在遇到情况时会发送事件 24
将事件与方法关联起来 25
你的iPhone工具箱 35

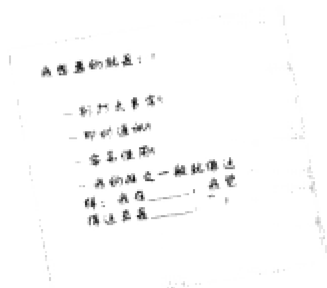
【拼吾爱-最新编程资源的分享下载站】

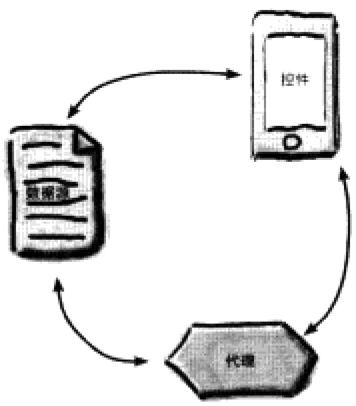
2 iPhone应用的模式

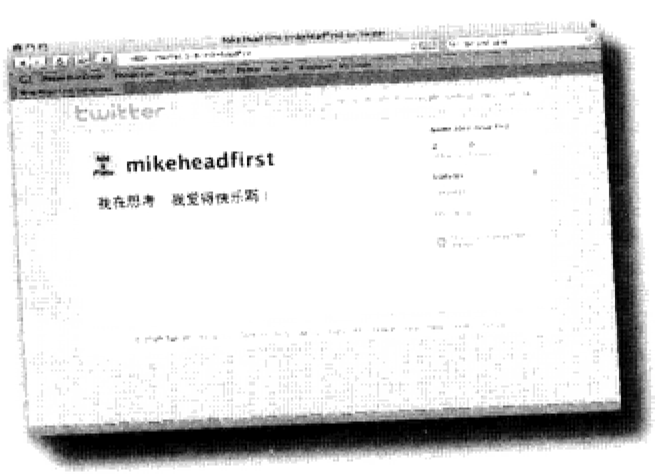
你好，@twitter!

应用都具有许多活动部件。好吧，实际上它们没有真正的活动部件，但是它们确实具有很多UI控件。一个典型的iPhone应用，可不会仅仅有一个按钮，现在是时候让我们来构建一个这样的应用了。这要使用一些更复杂的部件，意味着你也需要比以前更加注意应用的设计。在这一章里，你将学到一些用在iPhone SDK里的基本的设计模式，并学会如何组建出一个更大型的应用程序。









首先我们需要找出迈克（真正）的需求 39

应用之中，设计为王——iPhone人机界面指南 44

HIG关于选取器与按钮的指导意见 47

为InstaTwit（即推）创建一个基于视图的项目 48

根视图的生命历程 52

首先，从迈克那里获取数据 55

当你需要受控的输入时，就用选取器 56

将迈克的数据填入选取器的各行条目中 57

选取器从数据源获取它们的数据…… 58

有一个模式是为此准备的 59

首先，声明控制器遵循这两个协议 64

数据源协议有两个必需的方法 66

连接数据源与连接动作方法和输出口类似 67

代理协议中只有一个必需方法 68

按钮需要连接到事件 72

连接选取器和输出口 78

通过选取器的引用获取选中值 79

你的iPhone工具箱 87

【拼吾爱-最新编程资源的分享下载站】

3 面向iPhone的Objective-C教程

Twitter需要多姿多彩

在第2章中我们颇写了不少东西，但是用的那是什么语言啊？ 写的这些代码中，有些部分你可能有点眼熟，不过也是时候揭开谜底了，让你明白这到底是怎么回事了。iPhone SDK自带了一些非常棒的工具，让你不必事事都要写代码来实现，但是如果你不学习一下底层使用的语言，不了解属性、消息传递以及内存管理，还是无法编写一个完整的应用的。除非你能做到全部应用都只使用默认的组件！而你希望能做到更多的事情，对吧？



瑞妮有点明白过来了…… 90

给自定义输入腾地方 91

头文件为你的类描述接口 93

自动生成的访问方法也要进行内存管理 99

要让内存保持一致，你只需要记住两件事 101

但是当迈克输入完毕时…… 111

定制你的UITextField 113

使用键盘的组件会请求它出现…… 114

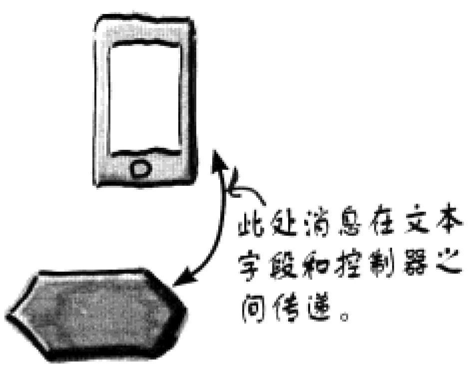
让文本字段让出焦点 115

Objective-C中的消息使用有名称的参数 117

用消息传递告诉视图控制器完成按钮何时被按下 118

还是有些不对劲 122

你的Objective-C工具箱 129





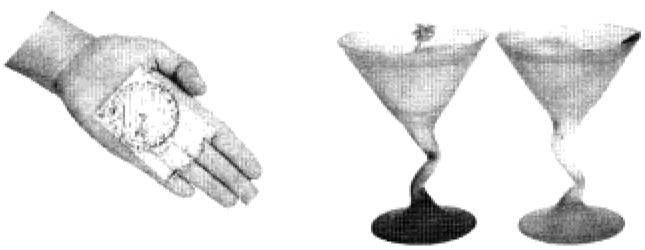
【拼吾爱-最新编程资源的分享下载站】

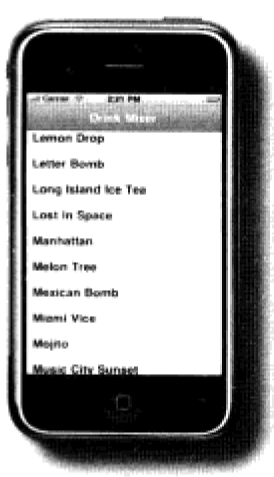
4 多视图

表格中的视图

大多数iPhone应用中的视图数目都不止一个。我们已经用单个视图编写了一个很酷的应用，但是用过iPhone的人都知道，大多数应用可不是像这样的。有一些更引人注目的iPhone应用使用多个视图，在处理复杂信息方面的工作卓有成效。我们打算先从导航控制器和表格视图开始，这两者就类似你在邮件应用与联系人应用所见到的那种界面。只不过我们会做一点小小的变化……







那么，怎么把视图结合在一起呢？ 135
导航模板会把多视图整合到一起 136
从表格视图开始，学习使用导航模板 137
表格就是一组单元格 140
再来几种饮料 148
Plist是一种保存和载入数据的简便方法 150
数组（及其他一些类型）内建了对plist的支持 153
用细节视图深入发掘数据 156
仔细考察细节视图 157
用导航控制器在视图之间切换 167
导航控制器维护了一个视图堆栈 168
字典将信息按照键－值对来存储 172
调试——iPhone开发之暗面 175
调试冒险之旅第一站：控制台 176
与你的应用程序在运行期间互动 177
Xcode在应用中断运行后也会为你提供支持 178
Xcode调试器会向你展示应用程序的状态 179
这到底是怎么回事？ 181
你的iPhone工具箱 183

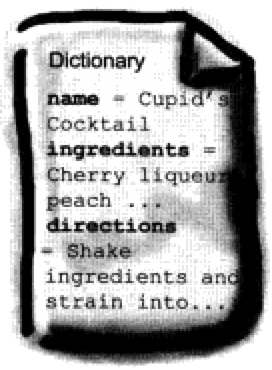
【拼吾爱-最新编程资源的分享下载站】

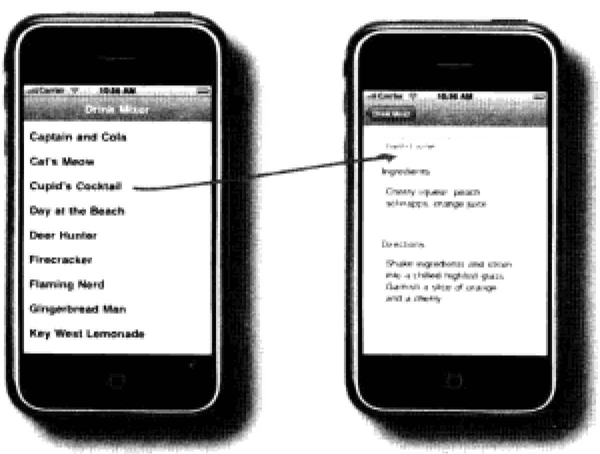
5 Plist与模式视图

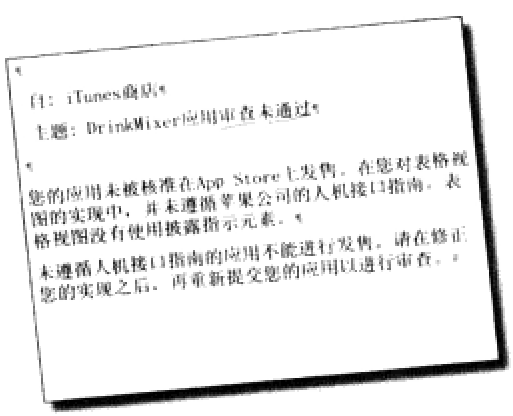
对应用千锤百炼

那么你的应用几乎可以工作了…… 每个应用都会经过这一步！首先会有一些功能可以使用，然后你决定加入一些其他的东西，又需要做一些重构，接下来又得对App Store中的一些反馈做响应。开发一个应用，从来都不会总是一个线性的过程，但是在这个过程中可以学到许多东西。








自：iTunes商店
主题：DrinkMixer应用审查未通过

您的应用未被核准在App Store上发售。在您对表格视图的实现中，并未遵循苹果公司的人机接口指南。表格视图没有使用披露指示元素。

未遵循人机接口指南的应用不能进行发售。请在修正您的实现之后，再重新提交您的应用以进行审查。

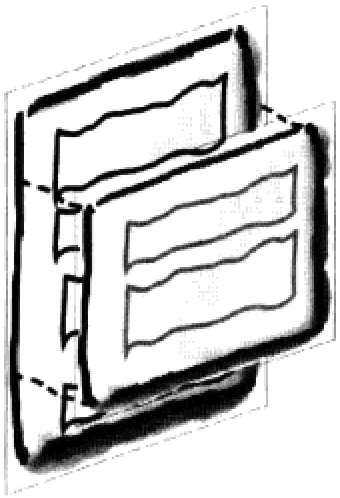
一切要从山姆说起…… 186
用调试器调查崩溃的罪魁祸首 188
让代码处理装有字典的plist 191
细节视图需要数据 194
每个字典中都有我们需要的全套信息 195
我们在可用性上出了问题 201
如果单元格会引导向更多信息，就用披露指示器 203
销售一度很强劲…… 206
用导航控制器按钮进行编辑 211
这个按钮应当创建一个新视图 215
我们需要一个视图……但并不一定是一个新的视图 216
视图控制器定义了视图的行为 217
Nib文件包含着UI组件与连接…… 218
你可以对视图进行派生和扩展，就像对其他类一样 219
模式视图让用户把注意力集中在手头的任务上…… 224
任何视图都可以作为模式视图呈现 225
我们的视图没有导航栏 230
创建保存和取消按钮 232
编写保存与取消动作方法 233
你的iPhone工具箱 237

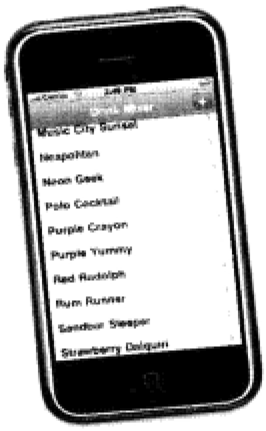
【拼吾爱-最新编程资源的分享下载站】

6 数据的保存、编辑与排序

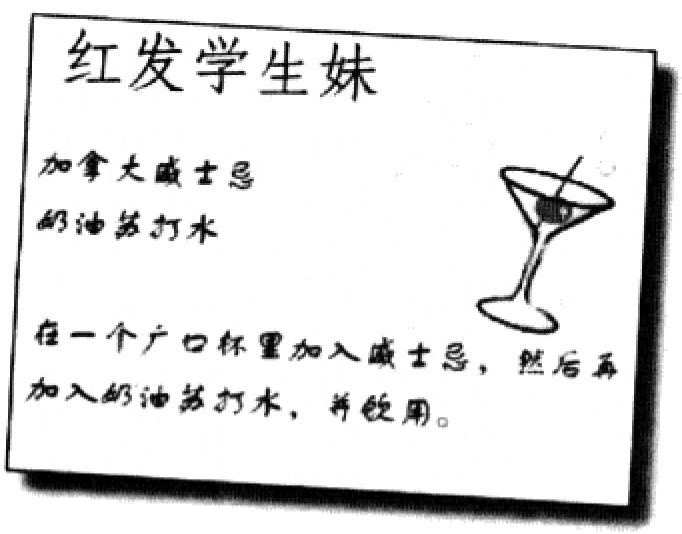
人人都能当编辑……

能显示数据挺不错的，但是添加和编辑信息才是让应用能够真正有价值的东西。 DrinkMixer很棒——它使用了一些单元格定制技术，而且使用plist字典来储存数据。它是一个很方便的参考工具应用，而你已经在添加新饮料方面开了一个好头。现在，是时候给用户以修改数据的能力了——保存、编辑以及排序——这会令这个应用对大家更有用处。在这一章中，我们会看一看iPhone应用中的编辑模式，以及如何通过导航控制器来引导用户。









山姆准备添加一种叫“红发学生妹”的饮料…… 240

……但是键盘总是碍事 241

我们需要将内容包装到一个滚动视图中 243

滚动视图大小与屏幕相同 245

键盘改变了可见区域 248

iPhone会通知你有关键盘的消息 250

向默认通知中心订阅事件 251

键盘事件会告诉你键盘的状态和尺寸 257

表格视图不知道数据发生变化 276

你得请求表格视图重新载入数据 276

数组的顺序也乱了 280

表格视图内建支持编辑与删除 288

你的iPhone开发工具箱 301

【拼吾爱-最新编程资源的分享下载站】

7 标签栏与Core Data

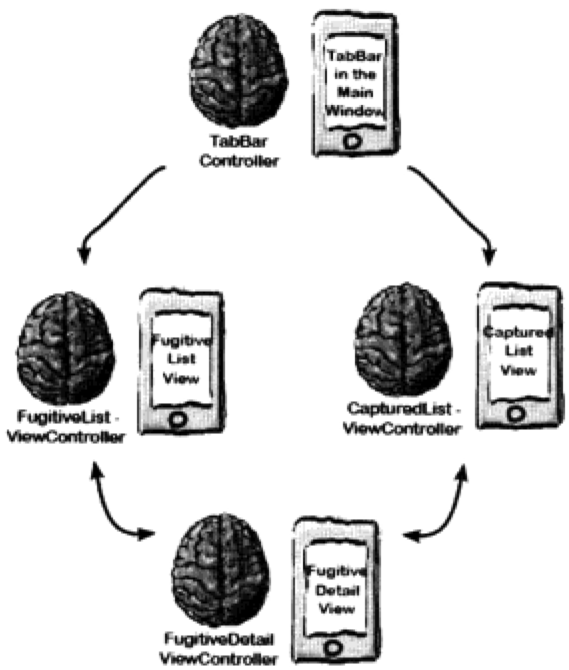
企业应用

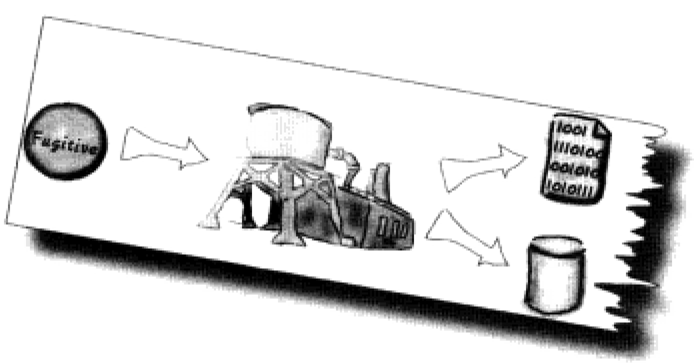
企业应用意味着以多种不同的方式来管理更多的数据。 对于iPhone应用来说，各种大大小小的公司是一个重要市场。一个小型手持设备再配以专门定制的应用，对于那些雇员始终处于移动状态的公司来说，其用途是巨大的。这样的应用大多数都会管理大量数据，而从iPhone 3.x开始，系统就内置了对Core Data的支持。通过使用这个框架，并辅以另一个控制器——标签栏控制器，我们就能够打造一个为正义而战的应用！





HF赏金捕猎 304
为iBountyHunter选择一个模板 308
画出iBountyHunter的工作过程…… 310
构建逃犯列表视图 316
接下来：被捕视图 318
在与鲍勃开了个短会之后…… 327
Core Data让你专注在应用上 329
Core Data需要知道如何进行加载 330
Core Data用一种受控对象模型来描述实体 333
打造你的Fugitive实体 334
不写一行代码，快速打造Fugitive类 341
用一个NSFetchRequest描述你的搜索 344
将数据库作为资源添加进来 354
模板设置为使用SQLite数据库 355
iPhone的应用程序结构决定了你能在哪些地方读和写 358
将数据库复制到正确的位置 359
未完待续…… 373
你的Core Data工具箱 375





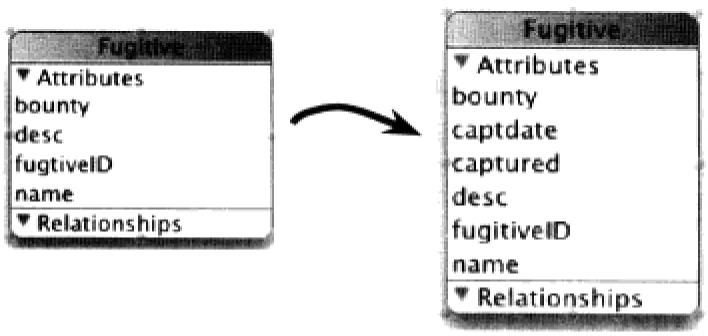
【拼吾爱-最新编程资源的分享下载站】

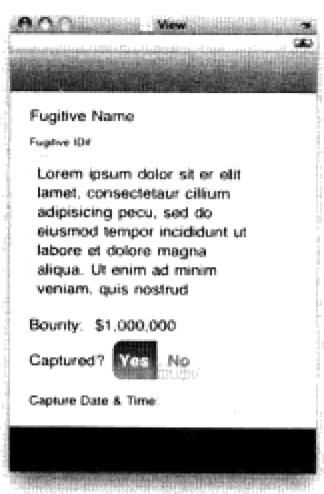
8 用Core Data迁移并优化数据

一切都改变

我们的应用工作得非常好。iBountyHunter成功地加载了鲍勃需要的数据，让他可以用一种方便的方式来查看逃犯。但是如果必须改变数据的话该怎么办？鲍勃想要一些新功能，这会对数据模型造成什么样的影响呢？在这一章中，你将学到如何处理对数据模型进行改变，以及如何利用更多的Core Data功能。







鲍勃需要一些文书工作 378
一切都源自我们的数据模型 381
数据还没有更新 384
数据迁移是一个常见问题 385
我们需要将旧数据迁移到新模型中 386
Xcode令数据模型版本控制变得轻松 387
Core Data可以“轻量”迁移数据 389
鲍勃有一些设计方面的建议 394
为鲍勃做一个快速演示 406
用谓词来过滤数据 408
我们需要在NSFetchRequest上设置一个谓词 409
Core Data控制器类提供了高效的结果处理 416
打造高效的“流线型”代码 417
下面我们需要修改查询，让它使用这个控制器…… 417
重构viewWillAppear使用结果控制器 418
我们需要刷新数据 423
你的数据工具箱 429

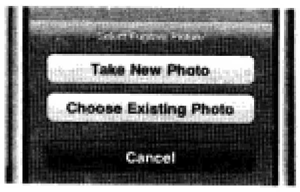
【拼吾爱-最新编程资源的分享下载站】

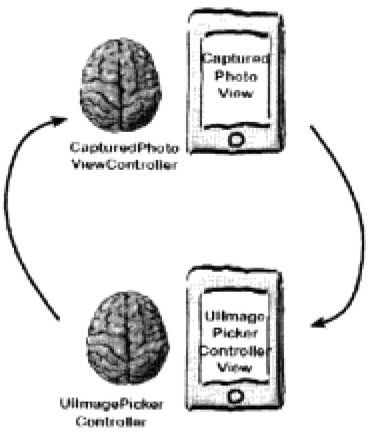
9 照相机、地图开发包和Core Location

现实世界中的证据

iPhone明晓它身处何方，见到了什么。 就像任何一个iPhone用户都知道的那样，iPhone的所作所为远不止管理数据那么简单：它还可以拍摄照片，找出你所在的位置，并整合这些信息让应用来使用。对这些功能进行整合的美妙之处在于，只要点触一下iPhone，进入它提供的工具，导入照片、位置和地图等功能就唾手可得，却根本不用编写多少代码。









对于鲍勃，收款还需要证据！ 432

通往照相机之路…… 441

有一个方法可以进行检查 451

用操作表单提示用户 452

除了时间之外，鲍勃还要知道地点 458

Core Location能以多种方式找到你 464

添加新框架 466

仅仅有经纬度对于鲍勃是不够的 472

地图开发包是iPhone 3.0中新加入的 473

对地图进行一点自定义设置 474

注释功能还需要一点雕琢 479

你的附加工具箱 485

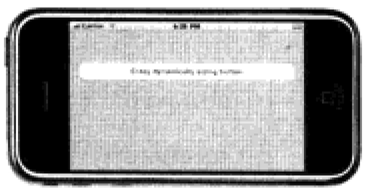
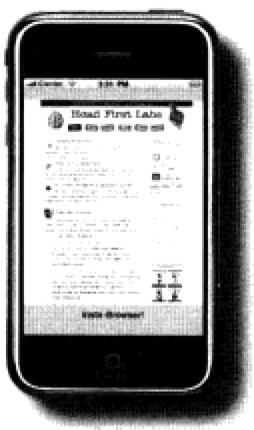
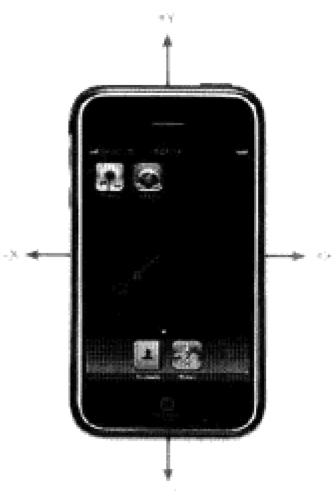
有你相伴真好！ 486

【拼吾爱-最新编程资源的分享下载站】

附录1 剩余

首要的六件事 (我们未涉及的)

是否觉得还缺了某些东西？我们了解你的感受……就在你以为结束了的时候，才发现还有些事没有做。我们离开你之前，必须再说明几项额外的内容，这些内容我们没法放进本书的其他地方。至少，如果你还想随身带着本书，而不是用一个带轮子的金属箱来盛放，这些内容就没法进入正文。所以请看一看，有没有什么是你还不会的？

#1. 国际化与本地化 488

对字符串资源进行本地化 490

#2. UIWebView 492

#3. 设备方向和视图旋转 494

处理视图旋转 495

用两个不同的视图来处理旋转 496

#4. 视图动画 497

#5. 加速计 498

理解设备加速 499

#6. 关于游戏的只言片语…… 500

Quartz和OpenGL 501

【拼吾爱-最新编程资源的分享下载站】

附录2 发布应用所需的预备工作

准备好进入App Store

你想让你的应用进入App Store，对吗？ 目前我们基本上还是在模拟器中使用我们的应用，这没问题。但是要想进入下一阶段，你还需要将一个应用安装在实际的iPhone或者iPod Touch上，然后才能申请将它放进App Store中。而要想做到这一点，唯一的办法就是成为苹果公司的注册开发者。即便如此，要把你写的应用放到自己的个人设备上，也不是在Xcode里按个按钮那么简单的事。为了这个目标，我们该和苹果公司谈谈了。

苹果公司指定了规则 504

设备备案将所有这些整合到一起 505

在Organizer中监视状态 506

【拼吾爱-最新编程资源的分享下载站】

1 新手上路

开始移动生活



iPhone改变了一切。 它是游戏平台，又是个人信息管理器，还是一个全功能网络浏览工具，哦，对了，还是电话。iPhone是很久以来出现的最令人激动的设备之一，而且随着App Store的开放，它也成了独立开发者的一个机会，令他们可以在世界范围内和那些大牌软件公司同场竞技。而发布你自己的应用程序所需的一切，仅仅是几个软件工具、一定的知识以及满腔热情。苹果公司提供了软件，而本书将带给你所需的知识；我们相信你一定准备好了满腔热情。

【拼吾爱-最新编程资源的分享下载站】

每个人都想要iPhone应用程序

App Store甚嚣尘上，其中充斥着大量的财富……

App Store是巨大的成功！

开发者已经并且正在向苹果公司的App Store提交的iPhone和iPod Touch的应用程序，不仅数目创下纪录，收入也极为可观……人不敢相

应

信

i



苹果公司的App Store庆祝周岁



在一年前，苹果公司启动了它那万众瞩目的App Store，并从此成为潮流的焦点、争议的中心以及许多许多财富的聚集之地。尽管许多应用程序是免费出售的，但也有许多并非如此，而且iPhone应用程序的价格持续飙升。

开发者们一直致力于用各种意想不到的方式来利用iPhone的功能，为这个你购自商店的设备带来更多乐趣以及便利。

甚至iPod Touch的用户，也可以将这些应用程序下载到他们的设备上。

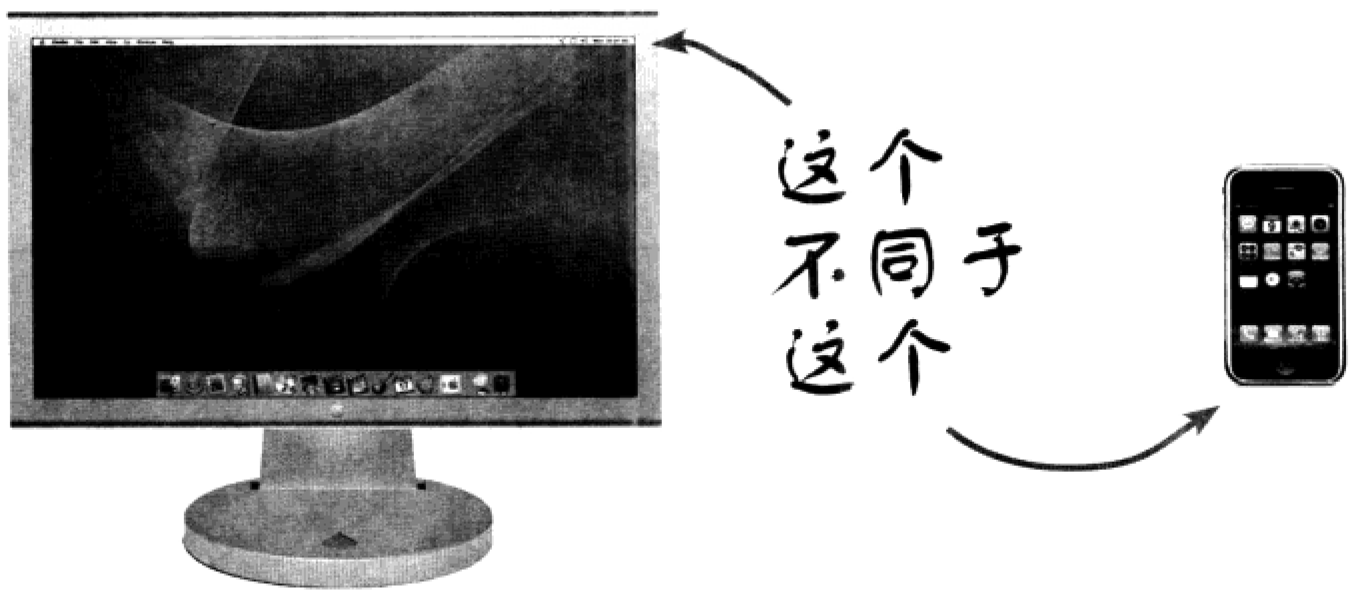
许多人每天都尝试几个新的应用程序，而你可以在你的电话上装满应用程序，足足占去好几个屏幕……

【拼吾爱-最新编程资源的分享下载站】

移动应用程序可不仅仅是移植桌面应用那么简单

有1万个理由值得加入App Store，而现在对于你来说正当其时。由此开始，你会学到如何设计并实现一个iPhone应用，但这并不同于为桌面系统进行开发，或者是编写一个网页应用程序。

对于iPhone应用程序来说，重要的是从一开始就要考虑周全。你需要不断地问自己："用户打算做什么？"去掉一切多余的东西，将用户需要提供的输入减至最低，并集中关注要点。





练习

选出在通常情况下，制作一个移动应用时需要注意的因素。

- ☐ 内存
- ☐ 使用费用
- ☐ 显示器功能
- ☐ 运行速度
- ☐ 访问互联网
- ☐ 用户输入工具（键盘、鼠标、显示器等）

这些因素中，哪些对于iPhone有所不同？ ....

....

【拼吾爱-最新编程资源的分享下载站】



练习答案

选出在通常情况下，制作一个移动应用时需要注意的因素。

移动设备内存有限，而且其中许多已预先分配。

- [x] 内存

某些移动用户发送短信或访问互联网都要花钱。

- [x] 使用费用

通常移动设备的显示尺寸和分辨率各不相同，你需要对此早作安排…

- [x] 显示器功能

- [x] 运行速度

移动设备的用户时间紧迫，因此应用需要快速加载与运行。

- [x] 访问互联网

对于移动用户，网络不是随时可用的，而且常常很慢。

- [x] 用户输入工具（键盘、鼠标、显示器等）

某些移动设备有方向键，某些有额外的按钮，还有的有触摸屏。

这些因素中，哪些对于iPhone有所不同？ 对于iPhone，通常数据访问和短信都不受限制，而且只需处理一种显示尺寸。还有就是没有键盘，是全触屏的！

iPhone应用绝不是缩小的桌面应用

有许多议论都说iPhone是一个可以随身携带的小电脑。这当然没错，但这并不意味着iPhone的应用就是缩小的桌面应用。下面列出的，是一些你在设计iPhone应用程序时可能遇到的最重要的问题。

- **iPhone的屏幕很小，而且要专注于任务**

  即使iPhone的屏幕可以说令人惊叹，但仍相对较小（320×480）。你得对每个画面都深思熟虑，令其专注于用户正在进行的特定任务。

- **iPhone的CPU和内存有限**

  在iPhone上并没有虚拟内存，而你使用的每一滴CPU资源都意味着更多的电力消耗。iPhone OS会密切监视系统，如果你的应用对内存消耗过大，会被直接杀掉。没人会想要这样。

- **同一时刻只能运行一个应用程序**

  如果是你的应用程序正在运行，为什么还要在乎这一点呢？那是因为一切皆有可能，比如电话铃响了，或者收到一条短消息，用户单击了一个链接等，你的应用会被关闭，用户会接着使用另一个应用程序。你得随时都能优雅地退出，并且可以在用户继续使用时，让他们回到一个合适的点。

【拼吾爱-最新编程资源的分享下载站】

iPhone应用剖析

在开始创建自己的首个应用之前，让我们先看看一个典型的iPhone应用程序是由哪些成分组成的。

首先是一个或几个视图……

iPhone应用是由一个或多个视图组成的——在一个通常的程序中，这些视图上还有其他GUI组件，比如文本字段、按钮、标签等。游戏也有视图，但是通常不会使用一般的GUI组件。游戏一般需要使用自己定制的界面，这些界面是用OpenGL或者Quartz这样的库创建的。



视图可以用代码来构建，或者用Interface Builder以图形方式构建，也可以二者结合。大多数应用使用混合方式。

然后是令视图工作的代码……

iPhone应用在GUI（视图）和实际提供应用程序逻辑的代码之间有明确的区分。通常来说，每个视图背后都有一个视图控制器，负责对按下按钮、选择表格中的某一行或倾斜电话之类的事件做出反应。这些代码几乎总是用Objective-C编写的，使用的工具则是苹果公司的IDE——Xcode。



Xcode是编写iPhone应用的IDE。其中含有许多应用程序模板，可以用来作为程序的基础。

...而其他任何资源，都会被打包进你的应用程序中。

如果你并不熟悉OS X开发，可能会惊讶地发现：应用程序（iPhone的以及完整版的桌面应用）实际上只是目录而已。任何应用目录都包含有实际的二进制可执行文件、一些关于应用程序的元数据（作者、图标文件名、代码签名等）以及应用程序的任何其他资源，如图像、应用程序数据、帮助文件等。iPhone应用程序的行为是一样的，所以你告知Xcode应用程序所需的其他资源，都会在构建应用程序时被打包在一起。



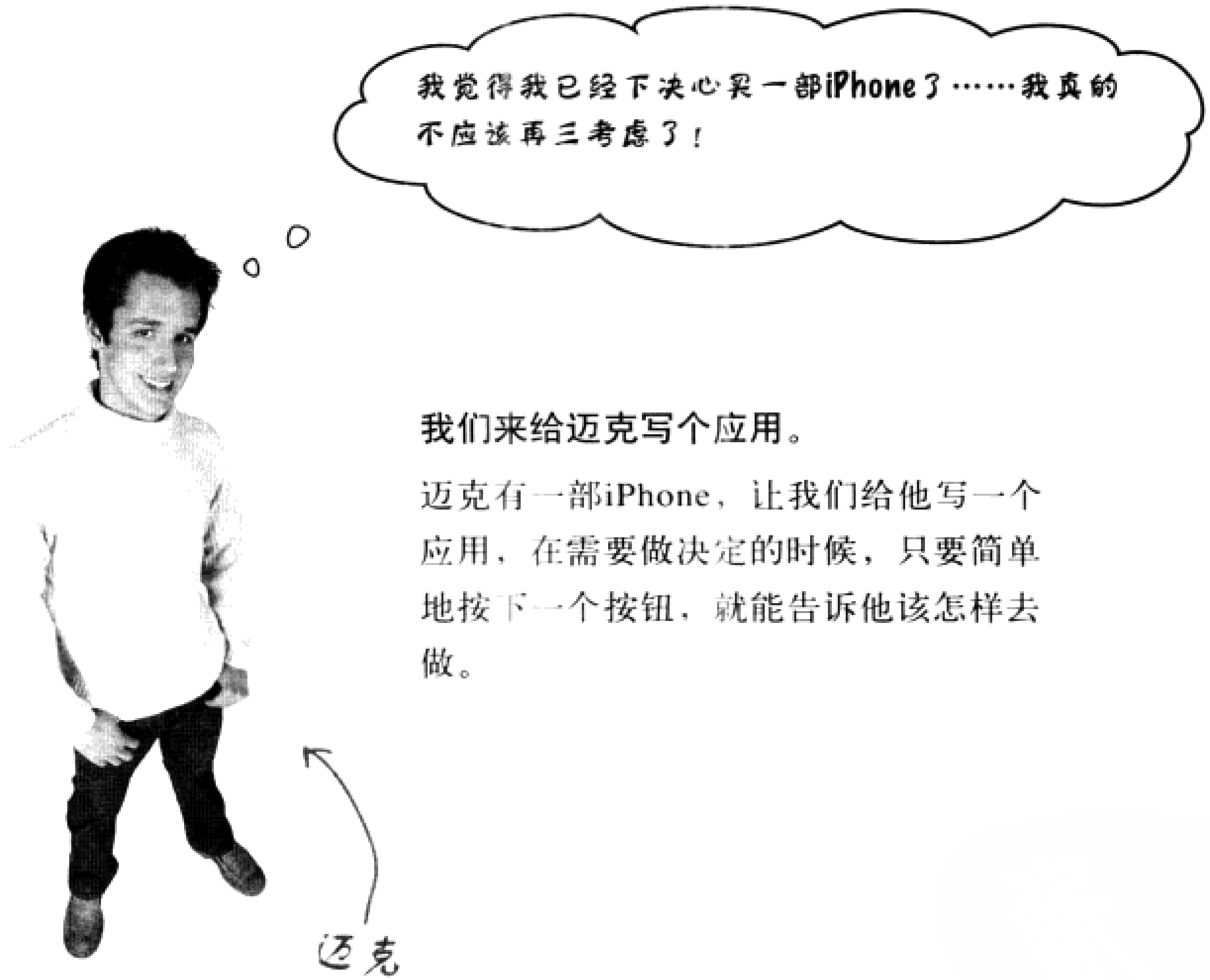
每个iPhone应用都有一些资源。至少，你的应用程序会有一个图标文件，一个关于应用程序自身信息的Info.plist文件，以及实际执行的二进制文件。其他常见资源还包括界面文件，也称为nib文件。

现在开始着手制作你的首个iPhone应用……

【拼吾爱-最新编程资源的分享下载站】

迈克有点左右为难

迈克这个人挺不错，但总搞不清自己想干什么。请帮助他搞清楚自己的目标，别再犹豫不决，给他一个爽快的回答。



我们来给迈克写个应用。

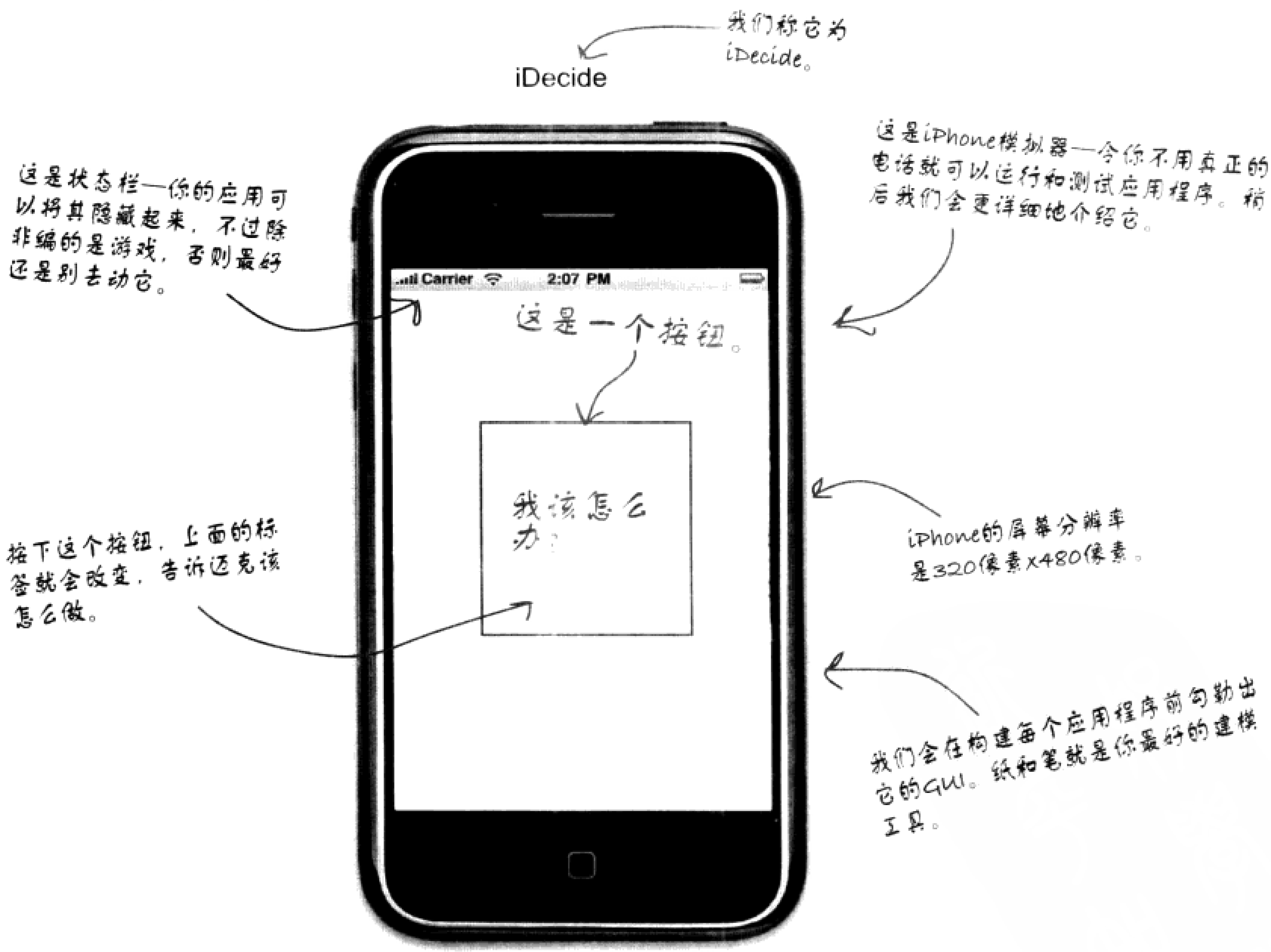
迈克有一部iPhone，让我们给他写一个应用，在需要做决定的时候，只要简单地按下一个按钮，就能告诉他该怎样去做。

【拼吾爱-最新编程资源的分享下载站】

第一眼留个好印象

当用户打开你的应用程序时，他们看到的第一样东西就是你的视图。它必须是可用的，并专注于你的应用程序的行动目标。在整个这本书中，只要我们开始写一个新程序，就会花一些时间来勾勒出我们所期待的应用的样子。

我们的第一个应用程序十分浅显：它就是一个单一视图，上面有一个按钮，迈克可以按下这个按钮，然后得到答案。为了简单起见，我们在他按下按钮后，就将按钮的标签换掉，来告诉他该做什么。



现在我们知道要构建什么了，开始了解工具吧。

【拼吾爱-最新编程资源的分享下载站】

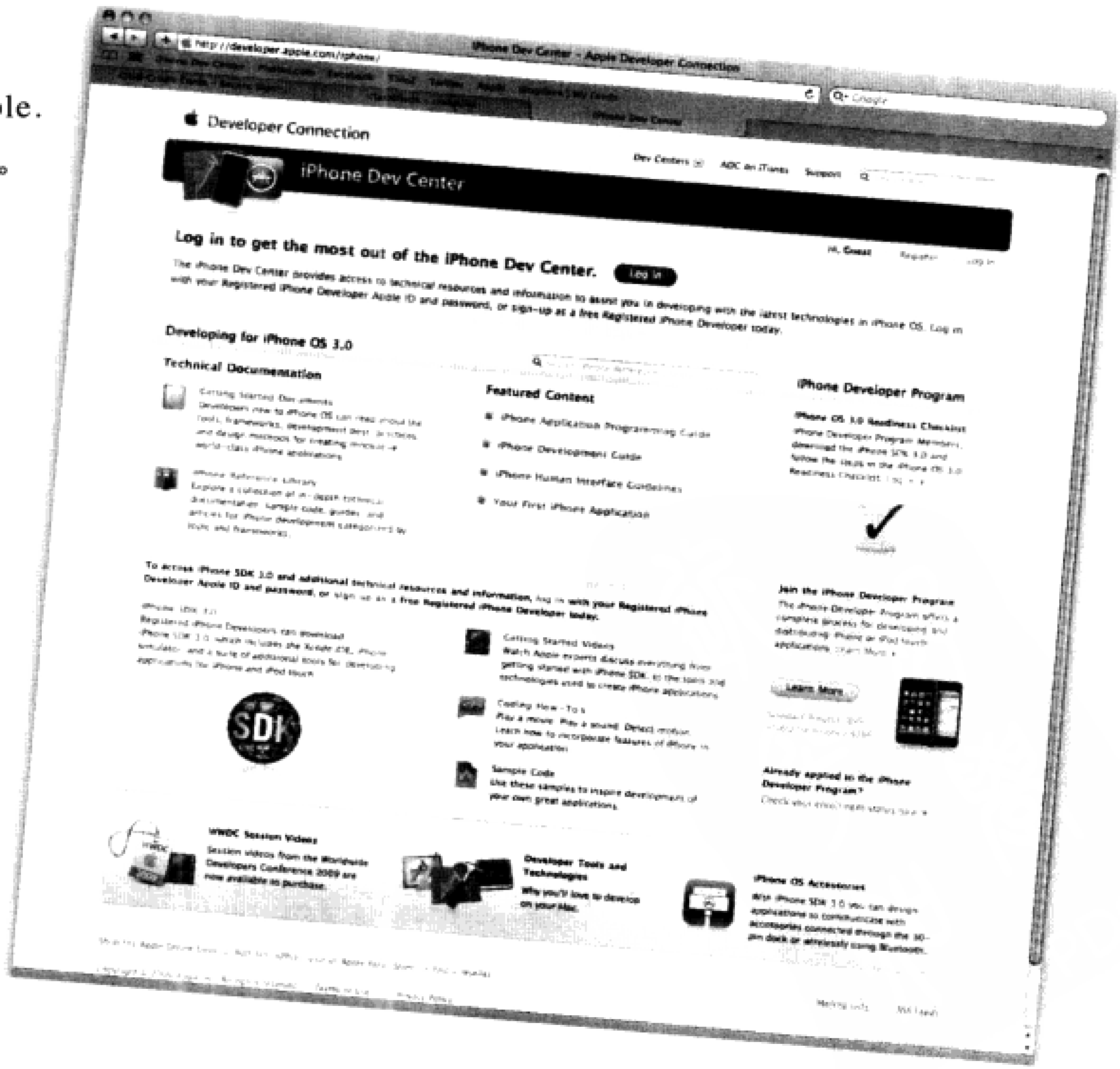
一切都从iPhone SDK开始

该是拿起工具的时候了。前往http://developer.apple.com/iphone，简单地注册之后，你就可以免费下载SDK（以及其他有用的苹果公司开发资源）了，不过要想在App Store中发布一个完整应用，或者将你的应用安装到iPhone上进行测试，还需要成为付费的标准或者企业开发者。SDK中带有一个模拟器，可以在你的Mac上直接进行测试，所以现在你只需要进行免费注册就可以了。

SDK中带有Xcode、Instruments、Interface Builder以及iPhone模拟器。在Xcode中你可以使用Objective-C为iPhone编写代码。Interface Builder用来以图形化的方式编辑界面，Instruments则帮助了解应用的内存使用以及性能等情况，而模拟器用于测试。

- [ ] 在网址http://developer.apple.com/iphone注册成为开发者。
- [ ] 下载最新的SDK；本书基于3.1版的SDK。请在页面顶部找到“Download”按钮。
- [ ] 安装SDK。安装完毕之后，你就可以在/Developer/Applications目录下找到Xcode.app。双击即可将其启动。

你很可能会想要把它安到Dock上——我们会经常用到它。



【拼吾爱-最新编程资源的分享下载站】

有问必答

问：开发移动应用时，要考虑的最重要的问题是什么？

答：在开发移动应用时，有两点要时刻牢记。首先，设备资源有限：内存、CPU、外部存储、网络访问速度等均是如此。其次，移动应用程序的使用模式不同。移动应用通常是便利型程序——用户要打开你的应用程序，迅速达到目的，然后回归现实世界继续进行之前的行动。

问：我以前做过移动平台上的开发，简直是乱七八糟。没有一样东西在不同设备之间是行为一致的，你不能预设屏幕尺寸，在不同的设备上甚至按钮数目都不一样！这次会好一点吗？

答：绝对和以前不一样！对于大多数情况来说，iPhone开发避免了你说的问题。iPhone的屏幕分辨率全都是320像素x480像素，都有加速计、唯一的Home键等。不过……

问：现在有好几种不同型号的iPhone吧，它们都是一样的吗？iPod Touch呢？

答：并不是所有的iPhone和iPod Touch设备都是完全一样的。例如，不是所有设备都有摄像头或者GPS。网络访问速度也随设备而不同，依赖于它们连接网络的方式是EDGE、3G还是Wi-Fi。令事态更复杂的是，与以前型号的iPhone相比，iPhone 3GS具有更快的处理器和更好的显卡。如果你使用的功能并不是在所有设备上都存在，你必须确保你的代码能够处理功能缺失的情况。苹果公司将会对此进行检测（例如，在第一代iPod Touch上使用摄像头），并且在你的应用程序无法正确处理这些设备时，会拒绝让它进入App Store。

问：iPhone使用的是什么语言？

答：iPhone上的应用程序通常使用Objective-C编写，这是一种面向对象语言，也用于Mac开发。不过，你也可以在iPhone上使用C甚至是C++。由于iPhone的GUI与核心框架库是用Objective-C编写的，大多数开发者使用Objective-C来编制程序；然而用C写的支持库也是很常见的。

问：我必须使用IDE吗？我确实是那种热衷于命令行方式的开发者。

答：从技术角度来说，不必，对于简单开发你不是非得使用Xcode IDE不可。然而，IDE令iPhone开发变得十分简单，你真得好好问问自己，是不是真的要抛开它，特别是要把程序部署到真实iPhone或者模拟器上进行测试的话，Xcode是必需的。本书使用Xcode IDE以及其他苹果公司的开发工具，如Interface Builder，而且我们鼓励大家在抛弃这些工具之前，至少尝试着使用一下它们。

问：我能把我写出的应用程序给朋友们用吗？

答：也能也不能。首先，如果你想要把一个应用程序放到任何人的实际设备上（也包括你自己的），都必须成为注册的苹果公司iPhone开发者。只要你注册好了，就可以注册一个设备，并将你的应用程序安装到上面。不过这并不是一种推出应用程序的好办法，而且苹果公司限制了你能够这样注册的设备数目。这种方法对测试应用程序很有帮助，不过要想把程序散发出去就不管用了。

一种更好的方法是将你的应用程序提交到iTunes App Store。你可以选择将应用程序免费发布，或者也可以收费，但是通过iTunes App Store发布程序，会让全世界的人都可以使用你的应用程序（而且也可能会赚点钱）。稍后本书会更详细地讨论如何发布应用。

问：我可以为iPhone开发一个应用，然后再把它重新构建在其他手机上吗？比如WindowsMobile、Android或者Blackberry？

答：简而言之，不行。当你为iPhone进行开发时，会使用苹果公司的iPhone框架，比如Cocoa Touch，还会使用Objective-C。这些在其他设备上都无法获得。

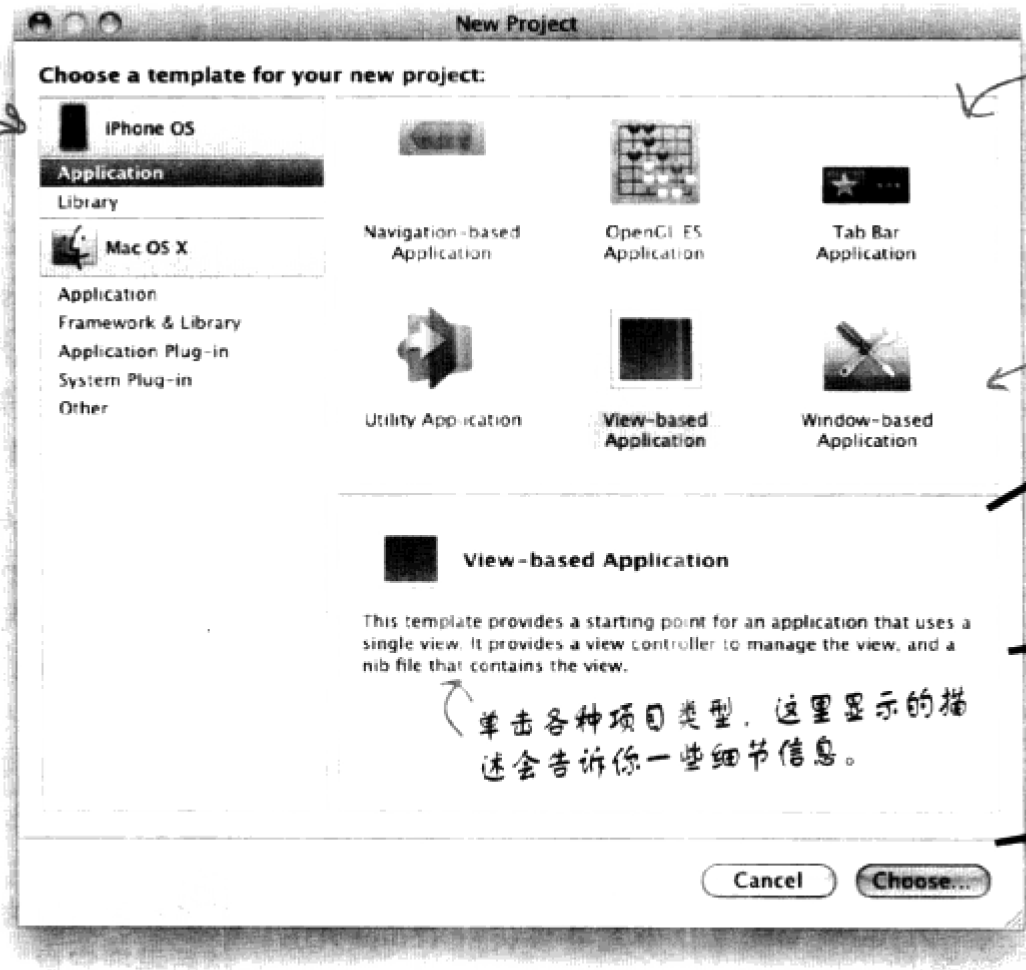
现在让我们开始。启动Xcode……

【拼吾爱-最新编程资源的分享下载站】

Xcode包含有应用模板，可以帮你入门

当你启动Xcode后，会看到一幅欢迎画面，在此你可以选择Create a New Project。就会看到如下的对话框。

这就是你用来进行Mac开发的同一个Xcode。由于我们现在为iPhone工作，请确保选中的是iPhone OS Application。

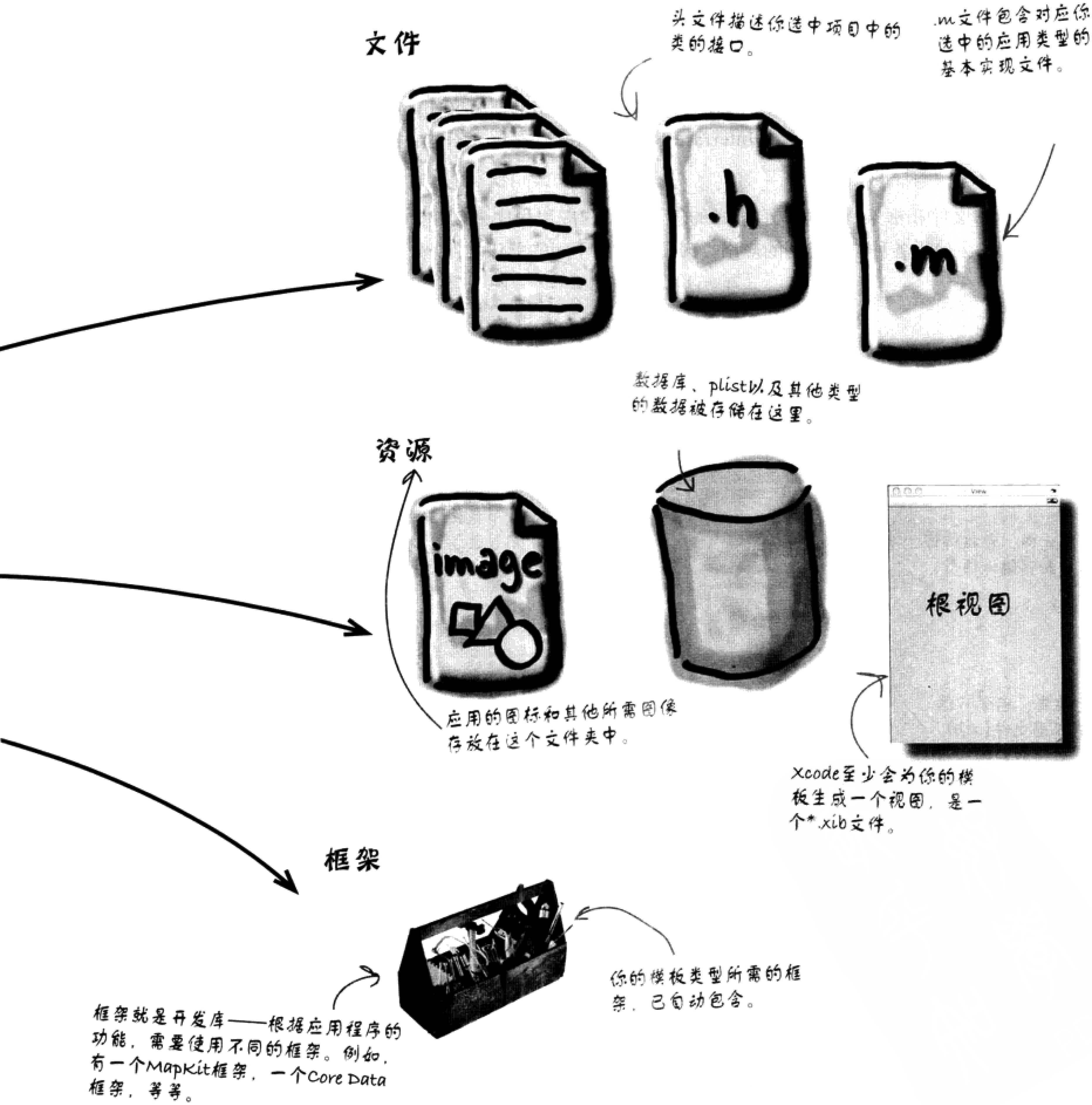


这些是基本的应用模板。基于你的选择，会为你生成不同的代码和文件。

在整本书当中，我们会使用不同类型的项目，并讨论对于每个应用为什么要选择某个模板而不是其他。对于iDecide来说，我们只有一个画面（或者说视图），也不准备加上翻页或者其他什么功能，所以用一个基于视图的应用程序来起步，并将其命名为iDecide。

Xcode模板中包含的，并不仅仅是源代码。

【拼吾爱-最新编程资源的分享下载站】

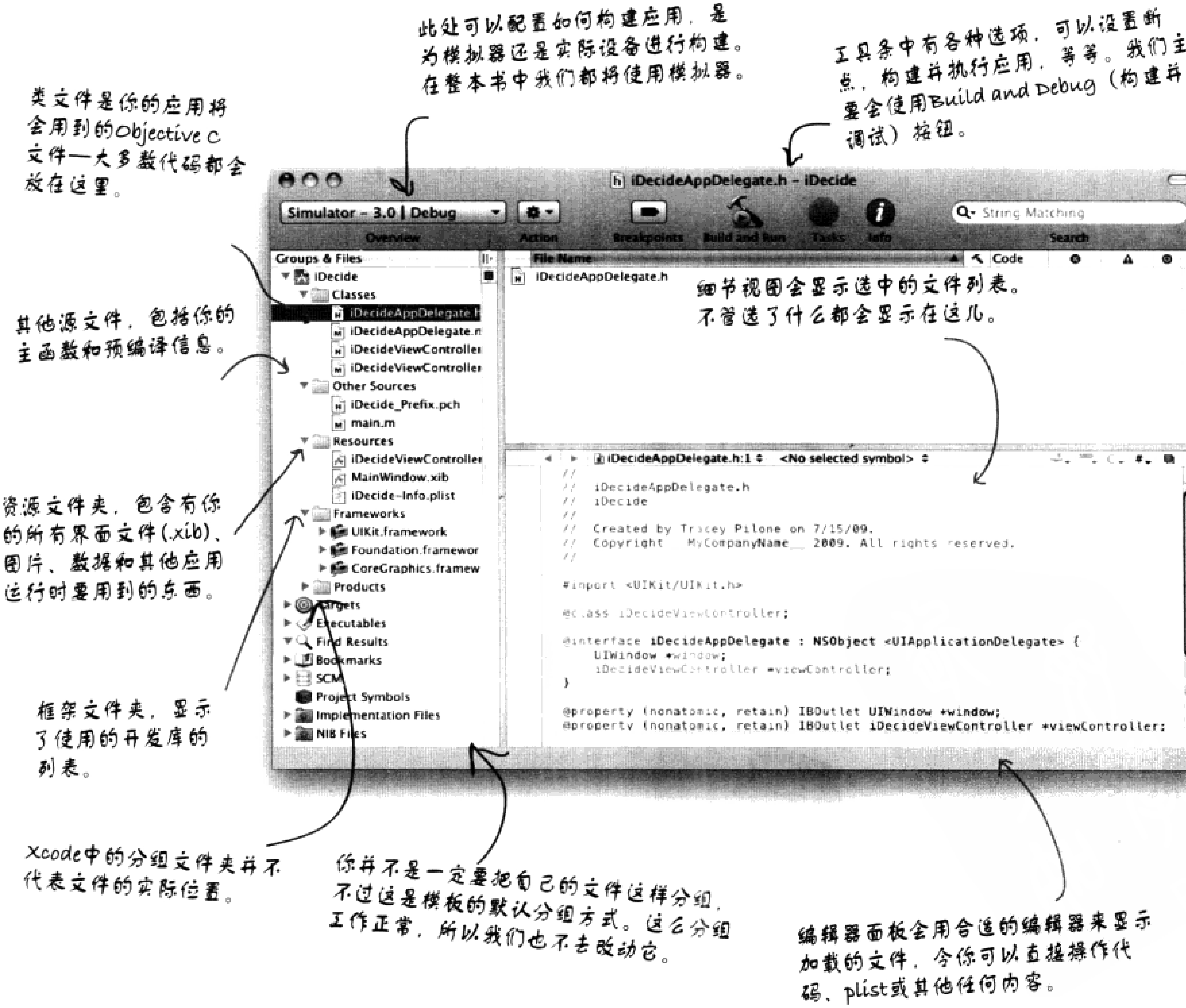


【拼吾爱-最新编程资源的分享下载站】

Xcode是iPhone项目的轴心……

当你用Xcode新建一个基于视图的项目时，它会生成下面列出的所有文件。我们也会使用其他一些SDK自带的工具（特别是Interface Builder和模拟器），不过它们也都是作用于这里所显示的文件的。

这里所显示的文件和框架是根据我们选择的基于视图的应用程序来创建的。在后面的例子里，我们会使用各种不同类型的应用，因此会得到不同的默认文件。



【拼吾爱-最新编程资源的分享下载站】

并在编写应用的过程中无处不在

Xcode并不仅仅是一个文本编辑器。正如你所见，Xcode包括一些模板，可以帮助你着手开发一个应用程序。根据应用程序的情况不同，你可以使用某个完整的模板或者只用模板的某一部分，不过你终归会用到其中一个。当你将基本的应用模板准备就绪后，就可以更进一步地使用Xcode，做更多工作了：

- **维护项目的资源**
  Xcode会为你的项目创建一个新目录，并将各种文件排布在子目录下。你不必固守这种默认布局，不过如果要对其进行调整，应该在Xcode内部进行。Xcode还内建了对版本控制工具的支持，比如Subversion等，所以可以用来签出项目和提交项目变更。

- **编辑代码和资源**
  你会用Xcode来编辑应用程序的代码，而且除了Objective-C之外，它还支持其他多种语言。Xcode还内建有多种资源文件的编辑器，比如plist（稍后我们会对其进行讲解）。对于Xcode不能直接处理的资源，比如UI定义文件（.xib），可以在Xcode中双击这些文件，就会打开相应的编辑器，对于.xib就是Interface Builder。有些类型的文件，比如图片，Xcode只能进行查看，或者像声音文件一样仅仅显示在列表中。

- **构建和测试应用程序**
  对于构建代码并生成可用的应用程序所必需的编译器，Xcode全部自带。当应用程序编译好之后，Xcode可以将其安装到iPhone模拟器或者真实设备上去。Xcode包含一个顶级调试器，同时具有图形和命令行两种界面，让你可以调试应用程序。你也可以启动类似Instruments的评测工具，来检查内存泄露或者性能问题。

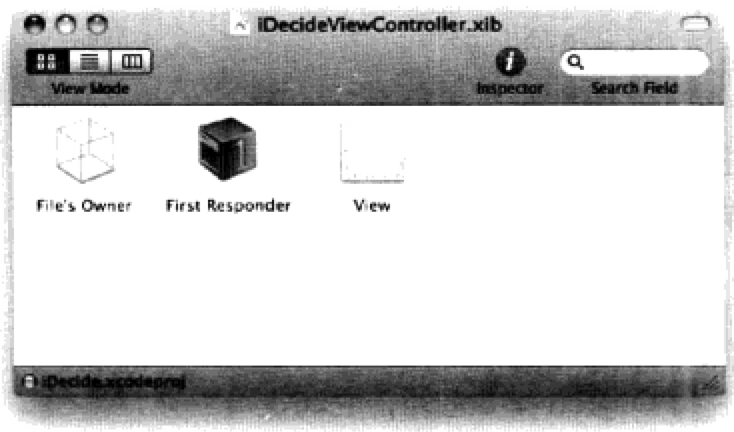
- **为销售应用程序作准备**
  当你详尽地测试过应用程序之后，就可以将其发售了，Xcode会对你的设备备案（provisioning profiles）和代码签名证书进行管理，让你可以将应用程序安放到真实设备上去，或者将其上传到iTunes App Store对外发售。

好，对于Xcode说得够多了：双击iDecideViewController.xib文件，让我们从视图开始。

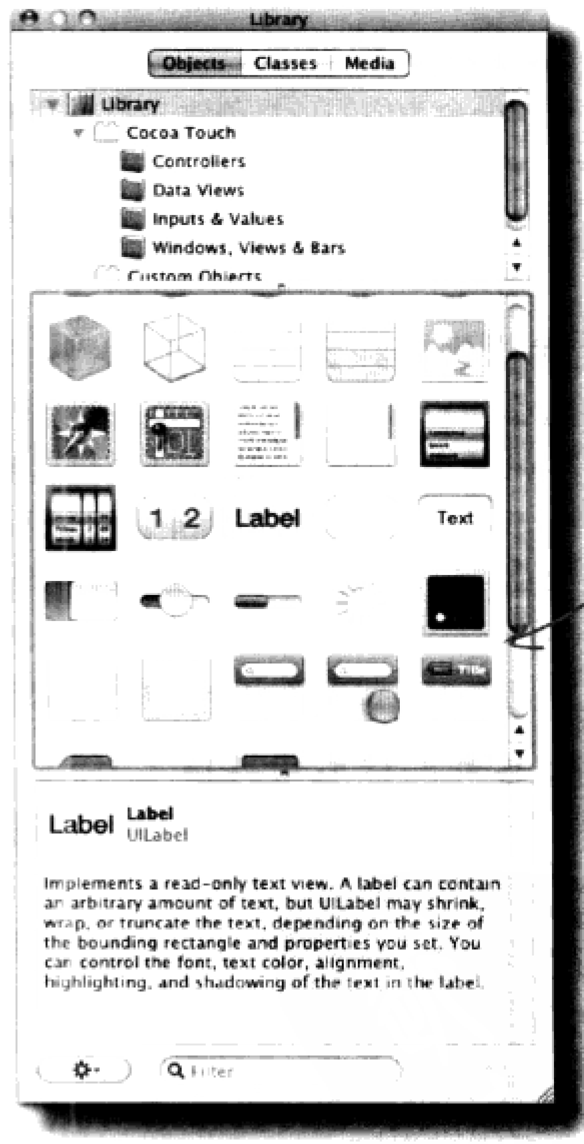
【拼吾爱-最新编程资源的分享下载站】

构筑界面，靠的是……Interface Builder

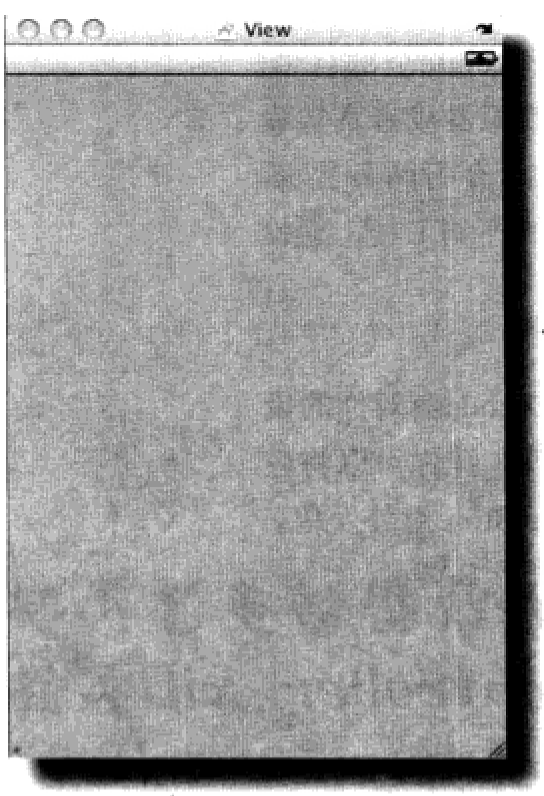
当你在Interface Builder中打开任何.xib文件时，都会自动显示出主窗口、视图以及一个UI元素库。Interface Builder允许你将任何基本的库元素拖曳到视图中，对其进行编辑，它还会处理代码和这些元素之间的联系。所有这些元素都来自于Cocoa Touch框架，这是专门为iPhone和iPod Touch定制的界面框架。



这是主窗口。其中显示了为当前特定nib文件创建的对象和视图。File's Owner和First Responder在每个nib中都存在，其他的内容会有所变化。稍后我们会非常详细地对此进行讲解。



库中显示了所有你可以选中放入视图中的元素。如果你上下滚动就会看到许多选项。



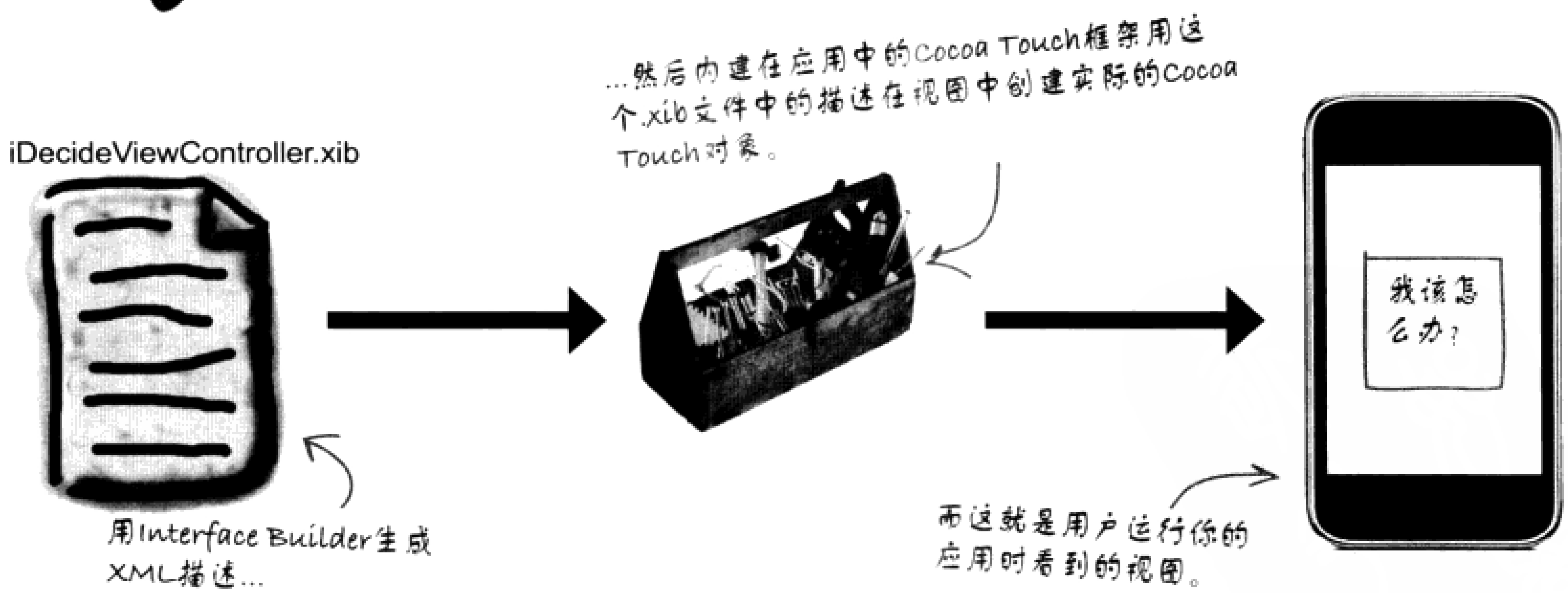
应用程序中的每幅画面都是一个视图。这里显示了在应用中的视图外观（除了任何需要加载的数据）。可以用代码构建视图，也可以用Interface Builder拖曳控件来完成。对于iDecide我们会使用Interface Builder。

GUI构建器看起来比较容易一些。我猜它会向我的文件里弄一些Objective-C代码进去吧？



不会 —— Interface Builder只创建nib文件。

Nib文件（扩展名为.xib）是XML文档，会在应用启动时被框架加载。在下一章中我们会好好讨论这个问题，不过现在重要的是了解Interface Builder并不会创建Objective-C代码。它只会为你构建的GU生成一个XML描述，Cocoa Touch框架会在运行时根据这个描述来为应用程序实际生成按钮什么的。在Interface Builder中所做的一切，都可以完全用Objective-C代码完成，不过你会发现，用GUI构建器来进行界面布局要容易得多。



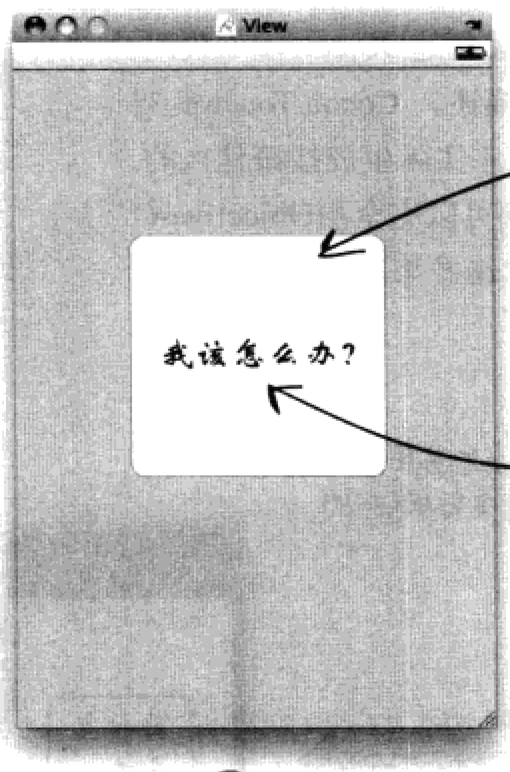
iPhone应用的视图叫做nib，而扩展名则是.xib。

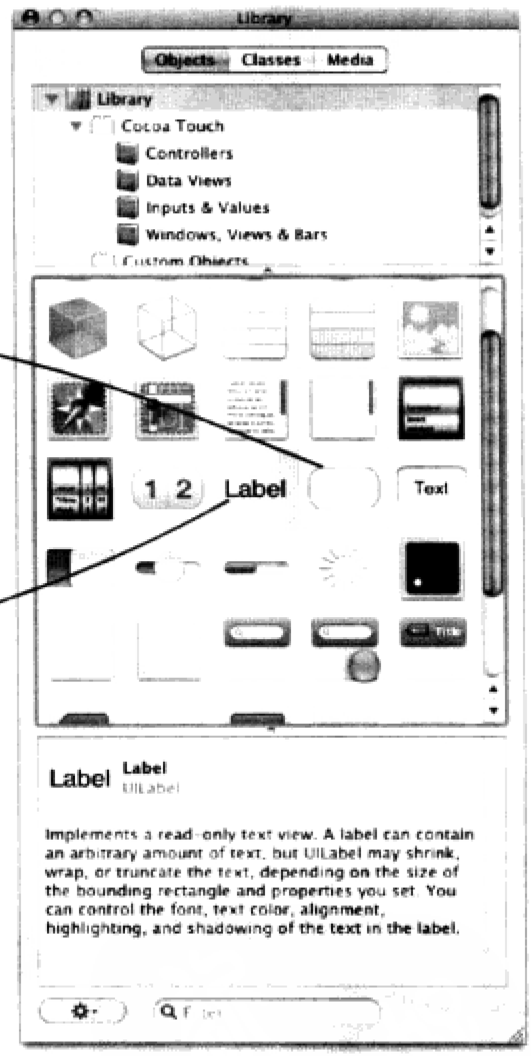
【拼吾爱-最新编程资源的分享下载站】

向视图中添加按钮

要在视图中加入元素，你只需要将想要的元素拖放到视图中即可。对于我们的应用，仅仅需要一个按钮和上面的标签。

❶ **将矩形的按钮拖入视图。**
按钮的初始尺寸会比较小，所以得让它变大一点。只要按住按钮的边角然后拉动，就可以改变它的尺寸。





❷ **将标签拖曳到按钮上。**
双击按钮上的新标签中的“label”字样，就可以编辑标签，输入新的文字，将内容改为：“我该怎么办？”再将其放置在按钮的中央。



运行测试

现在于Interface Builder中保存文件，返回Xcode并单击Build and Run（构建并运行），从Build菜单中选择或者按下主窗口中的按钮均可。这样就会启动模拟器。

【拼吾爱-最新编程资源的分享下载站】

iPhone模拟器让你可以在Mac上测试应用



模拟器是一个特别棒的用来快速测试应用的工具，而且是免费的。它里面并没有带上真正电话中的全部应用程序，不过行为基本上还是和真机一样的。当你打开模拟器时，你首先会看到Springboard（主界面），就像一个真正的iPhone一样，iDecide也已经安装好了（图标是默认的，稍后可以更换）。然后Xcode会打开应用，你的代码就开始运行了。

使用模拟器和iPhone有一些差异。首先，摇晃和旋转你的Mac并不会起任何作用。要模拟旋转或检查横向纵向视图切换，可以用Hardware菜单下面的一些命令来完成。



模拟器的限制。

用模拟器无法可靠地测试内存、性能、照相机、GPS以及其他一些功能。稍后我们会仔细谈一谈这些问题，不过内存使用情况和性能之所以很难在模拟器上测试，是因为Mac的资源与iPhone相比实在是太多了。要进行这些方面的测试，你得把应用安装到实际的iPhone上（这就意味着要在付费的开发者计划中选择一种，并加入其中）。

有问必答

问：还有其他东西在模拟器上不能用吗？

答： 模拟器上只能使用某些特定触摸手势，网络访问能力和定位功能也受到限制，而且没有加速计和摄像头。要了解更多信息，请参看苹果公司的iPhone OS 3.0库文档，在模拟器的Help菜单上就有。

用模拟器来入门学习编写应用程序很不错，但早晚你会不得不使用真实设备。此外，要知道iPod Touch和iPhone是两种不同的设备，能力也不一样。你最好在二者上面都进行测试，也就意味着你得加入一种付费计划。

问：Nib文件的扩展名是.xib，这到底是怎么回事？

答： 这是一种怪异的人为产物，反映了OS X的起源。Nib的历史要追溯NeXT被苹果公司收购之前的NeXTStep年代。在OS X Leopard中，苹果公司发布了一种新的nib文件格式，是基于XML模式的，扩展名是xib。所以如果扩展名是.xib，文件格式就是XML，然而人们仍然将其称为nib。在库中的类名称上你会发现更多的NeXTStep遗迹——几乎一切都是以“NS”开头的，也就是NeXTStep的缩写。

问：为什么我在模拟器中单击按钮，却什么事也没发生？

答： 由于Xcode帮我们做了这么多事，所以你很自然的会认为按钮会自动工作。不过，你想想我们才做了哪些事，创建了一些XML文件来让一个框架加载以便绘制一个按钮，但是我们还没告诉它该用这个按钮做些什么呢……

【拼吾爱-最新编程资源的分享下载站】

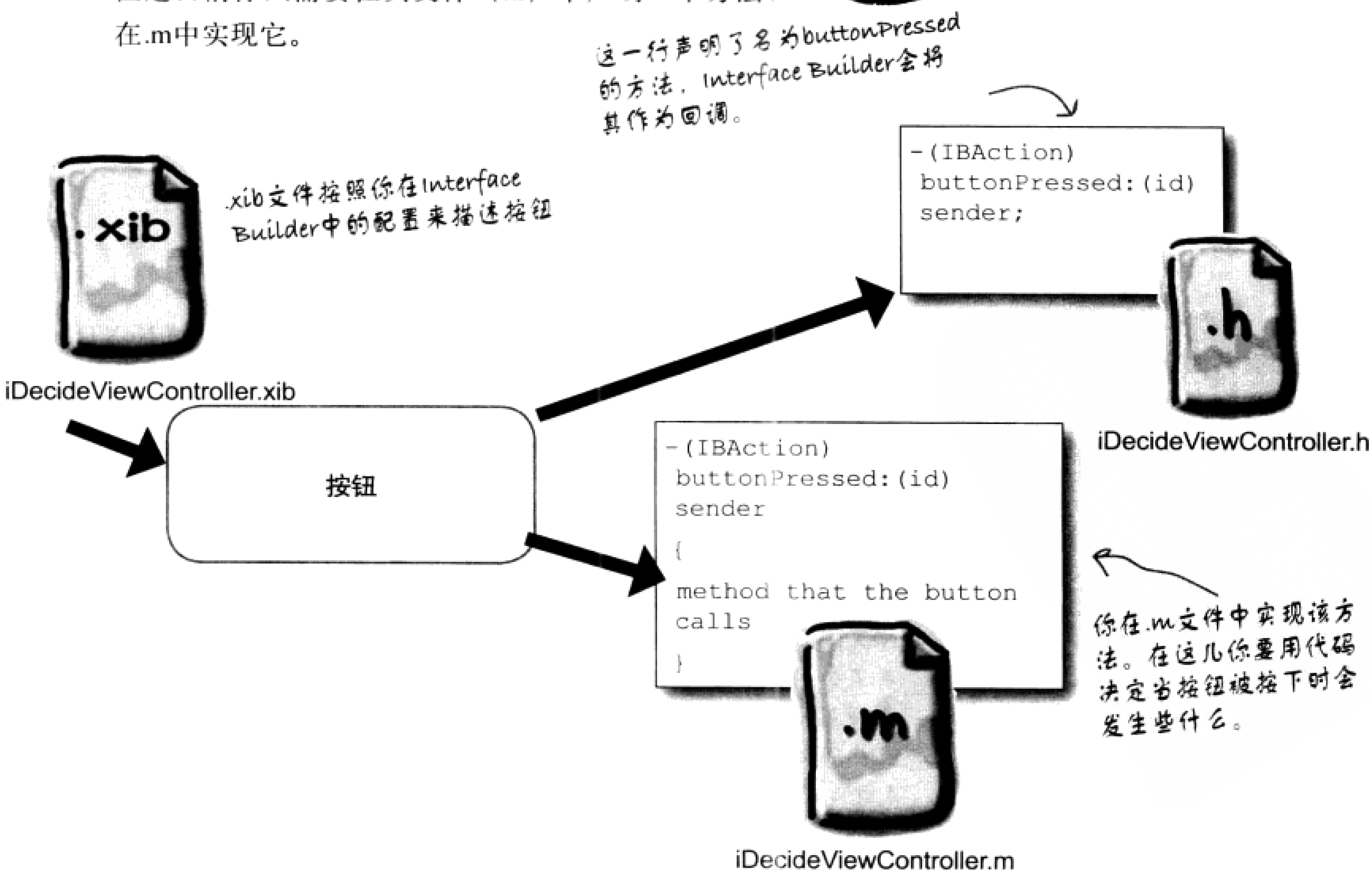
好吧，Interface Builder创建了XML，但我们还得写代码来实现按下按钮的功能，是这样吧？



UI的行为是用Objective-C实现的。

Interface Builder为你创建了按钮，但是要让按钮真正干点事，你还需要用代码控制它该怎么做。

控件在有情况发生时，比如按钮被按下或者文本字段中的文字发生了变化等，就会触发Objective-C事件。对于类似按下按钮这样的事件，Interface Builder必须把视图控件与控制器类中的动作方法关联起来，这些方法被打上了IBAction标签（表示Interface Builder动作）。后面我们会讨论Objective-C的语法，但是目前你只需要在头文件（.h）中声明一个方法，在.m中实现它。



【拼吾爱-最新编程资源的分享下载站】

拿起笔来

下面是按钮被单击时的处理代码。请将粗体字部分的代码加入iDecideViewController.h和iDecideViewController.m文件中。

```
#import <UIKit/UIKit.h>
@interface iDecideViewController : UIViewController {
    IBOutlet UILabel *decisionText;
}
@property (retain, nonatomic) UILabel *decisionText;

-(IBAction)buttonPressed:(id)sender;
@end
```

我们得改变标签文字，好告诉迈克答案，所以我们需要得到框架根据我们的nib文件构建出来的标签控件。

本书后面会对属性做更详细的讲解。

这是按下按钮时会被调用的动作方法。

iDecideViewController.h

```
#import "iDecideViewController.h"
@implementation iDecideViewController
@synthesize decisionText;

-(IBAction)buttonPressed:(id)sender
{
    decisionText.text = @"勇往直前！";
}
```

@synthesize告诉编译器，创建我们在头文件中声明的属性。

这是按钮按下时调用的方法的实现。

我们会通过标签的引用来改变它的文字。

```
- (void)dealloc {
    [decisionText release];
    [super dealloc];
}
```

在dealloc方法中可以清理使用过的内存。在第3章中会对其做详细讲解。

iDecideViewController.m

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

这是前面的代码，放置在iDecideViewController.h和iDecideViewController.m文件的完整上下文当中。

```
#import <UIKit/UIKit.h>
@interface iDecideViewController : UIViewController {
    IBOutlet UILabel *decisionText;
}
@property (retain, nonatomic) UILabel *decisionText;
-(IBAction)buttonPressed:(id) sender;
@end
```

iDecideViewController.h

这些是通常出现在头文件中的代码。包括新声明的IBOutlet和IBAction，以及为UILabel定义的属性。

当按钮被单击时，IBAction会处理一切，而IBOutlet则放在引用之前，指向我们为按钮输出文字的标签。稍后会对它们做更为详细的研究。

```
#import "iDecideViewController.h"
@implementation iDecideViewController
@synthesize decisionText;
-(IBAction)buttonPressed:(id) sender
{
decisionText.text = @"勇往直前！";
}

-    (void) dealloc {
    [decisionText release];
  [super dealloc];
}
```

iDecideViewController.m

这是实现代码。在此对按钮按下时调用的方法做出定义。我们使用了一个常量字符串来修改标签中的文字。请记住，decisionText是对我们在Interface Builder中创建的UILabel的引用。

调用release进行内存管理。Objective-C使用引用计数来管理内存（稍后详解之），要release之后才能真正释放内存。

【拼吾爱-最新编程资源的分享下载站】

运行测试

再次构建并运行代码。试着单击按钮，看看它是否已经可以工作了。



什么也没发生！

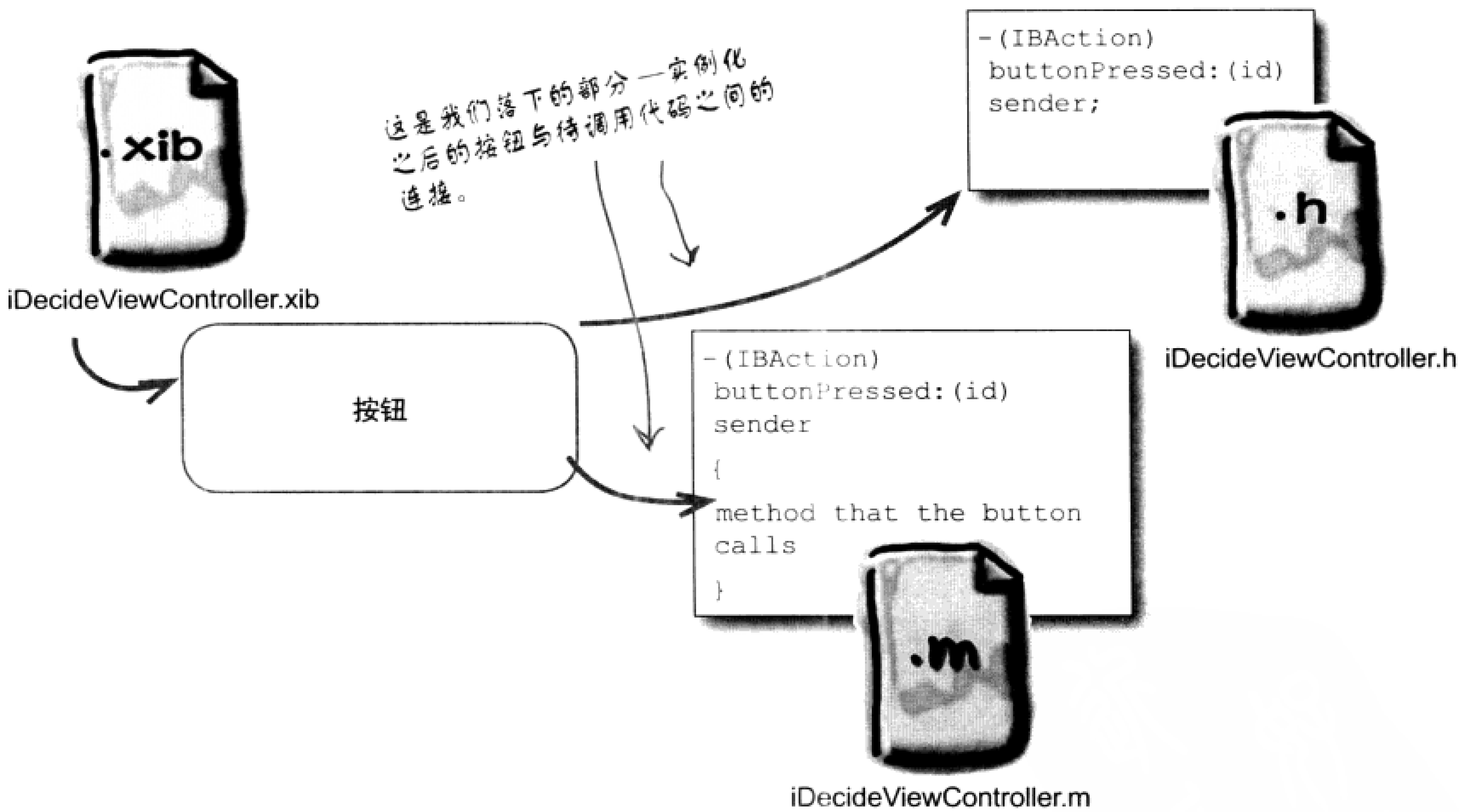
脑力练习

按钮怎么没变化啊？代码已经正确编译了，所以问题在于……

【拼吾爱-最新编程资源的分享下载站】

怎么回事？

已经安排好了Objective-C代码去处理按钮按下的情况，但是Interface Builder还不清楚它需要将按钮和这些代码关联起来。我们可以用Interface Builder将按钮与刚写好的buttonPressed方法挂接在一起。然后，当框架加载.xib文件时，就会将创建的按钮对象和我们的代码联系起来。



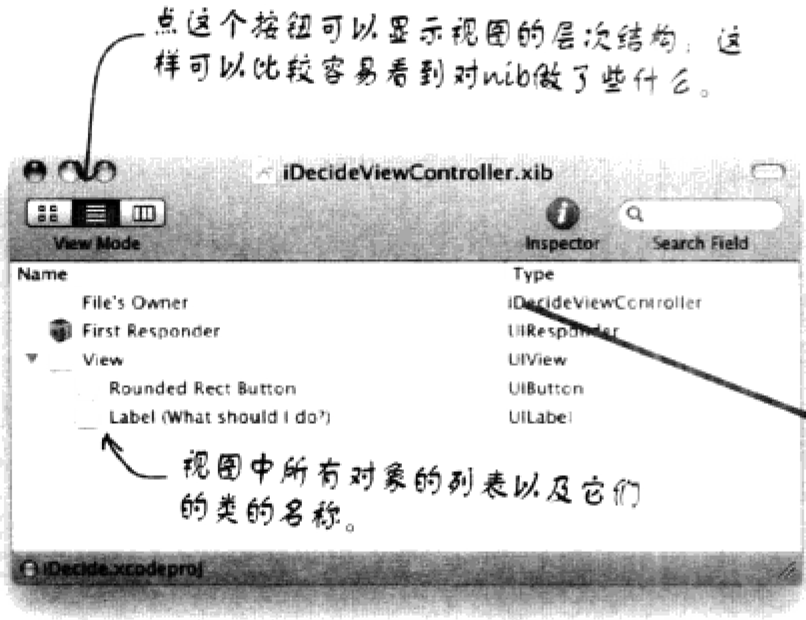
除非UI组件已经与代码关联，否则不会发生任何动作。

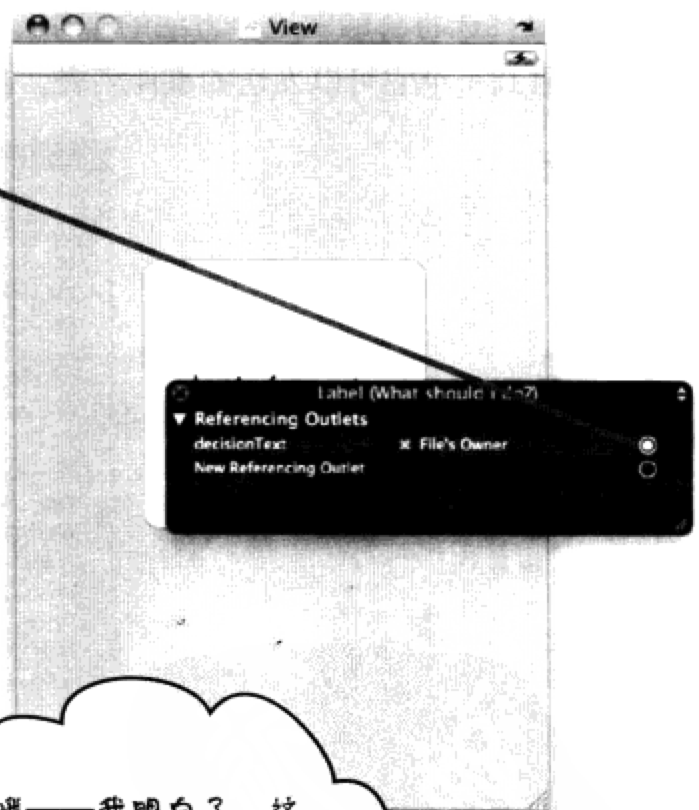
我们需要将代码的按下事件与我们的buttonPressed动作方法关联起来。这样我们的方法就会在用户单击按钮时被调用。之后我们要得到框架在加载nib时创建的UILabel的引用——这就是为什么要使用IBOutlet的原因。通过这个输出口（Outlet），我们就可以在单击按钮时改变UILabel的文字。

【拼吾爱-最新编程资源的分享下载站】

用Interface Builder将UI控件与代码关联起来

转回头再看Interface Builder中的iDecideViewController.xib文件，让我们把组件和我们新加的代码挂接在一起。





如果你的鼠标只有一个按键，可以按住CTRL键再单击。

❶ 右键单击你放在按钮上的标签，就会显示出一个事件与引用的列表。

❷ 单击New Referencing Outlet旁边的圆圈并拖曳到File’s Owner（代表将加载此nib的类文件，对我们来说就是iDecideViewController）。然后再单击decisionText输出口。这样一来当decisionText标签被生成时，我们的decisionText属性就会设置为对此控件的引用，这全靠有了IBOutlet标记。



【拼吾爱-最新编程资源的分享下载站】

Interface Builder会列出组件可以触发哪些事件

我们需要把正确的组件事件附着到代码上。我们之前写的动作方法，就可以和按钮关联起来：

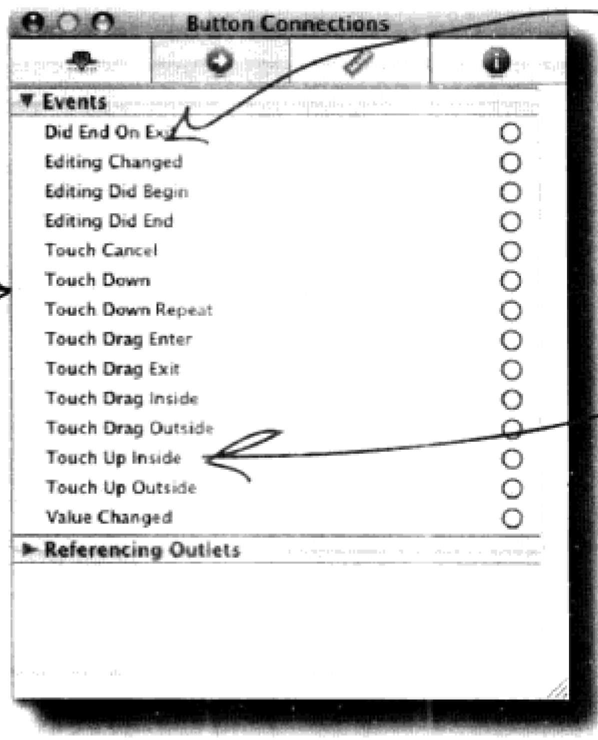
```
- (IBAction) buttonPressed:(id)sender;
```

所有的IBAction消息都接收一个参数：消息发送者。也就是触发事件的UI元素。

这是待调用的方法名。此方法可以任意取名，不过必须有一个类型为id的参数。

IB = Interface Builder

现在我们要挑选将会触发这个方法的事件。如果你右键单击Interface Builder中的按钮，就会看到一个它可以发送的事件列表。我们要选择TouchUpInside事件。



大多数事件都可以顾名思义。

这个列表显示了此按钮可以注册的全部事件。稍后本书会进一步介绍各种事件。

我们要使用“触摸于内部抬起”事件。

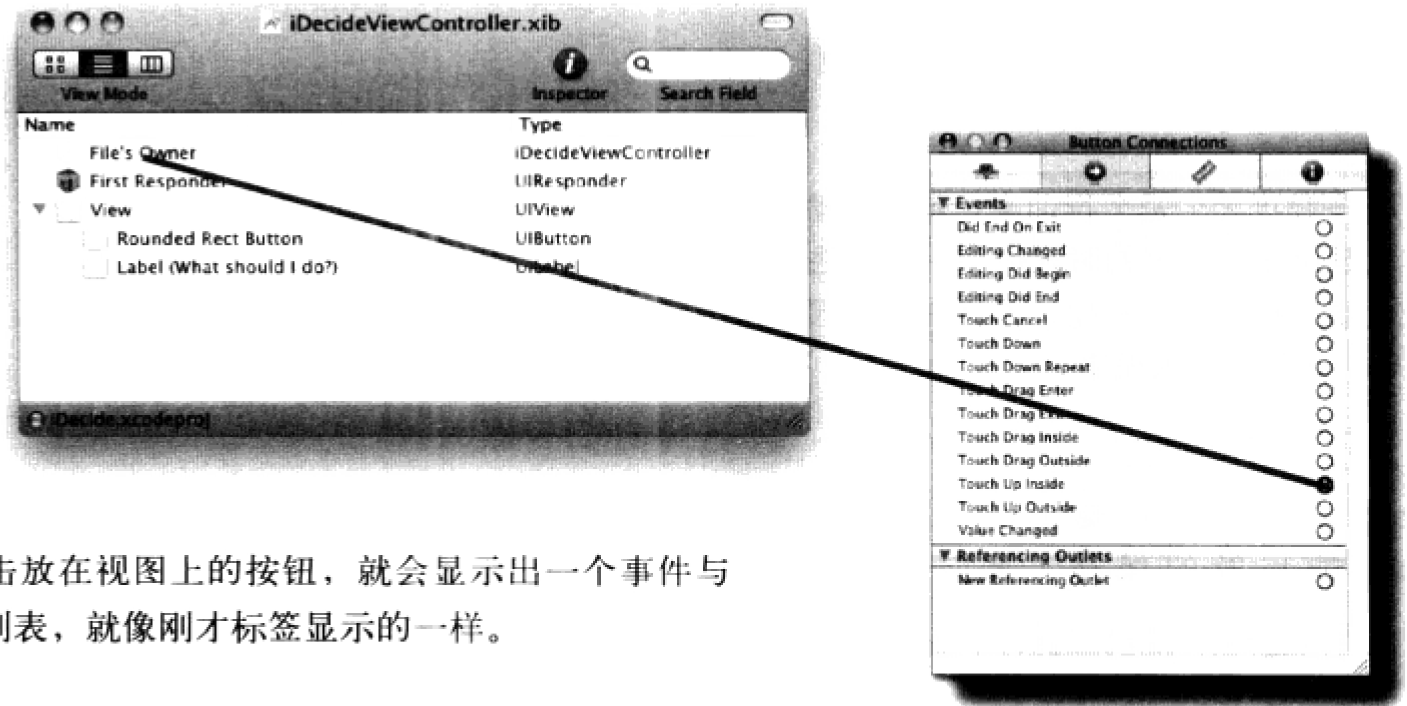
UI元素在遇到情况时会发送事件

只要元素遇到了情况，比如按钮被单击，就会发送一个或多个事件。我们要做的就是告诉按钮在事件发生时通知我们。我们会使用TouchUpInside事件。回想一下在iPhone上单击按钮的情形，其实是否单击在按钮内部并不重要，当你挪开手指时（触摸结束）实际的单击才发生。把事件和方法关联起来，就像关联元素与输出口一样。

【拼吾爱-最新编程资源的分享下载站】

将事件与方法关联起来

就像输出口一样，你可以从按钮事件向 File's Owner上拖曳关联关系，然后再选择应该调用的方法。



1. 右键单击放在视图上的按钮，就会显示出一个事件与引用的列表，就像刚才标签显示的一样。

2. 接下来单击Touch Up Inside旁边的圆圈，然后拖曳到File's Owner上。再单击buttonPressed动作。这样，当按钮被按下时，我们的buttonPressed方法就会被调用。

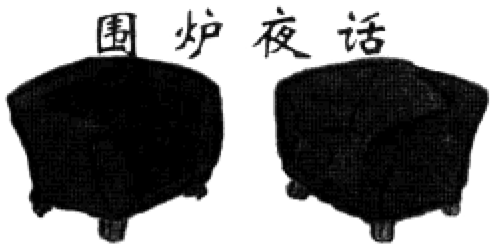


那我用IBOutlet还是IBAction要紧吗？反正Interface Builder也都能用？

这太要紧了！

它们俩可不一样。用IBOutlet表示你需要一个对界面中的某物的引用（例如，这样你就可以改变标签文字）。用IBAction表示你需要一个控件告知你的代码发生了一些情况（比如按钮被按下了）。

【拼吾爱-最新编程资源的分享下载站】



今夜话题：IBAction扬眉吐气了，盖过了好多位。

IBAction: 嗨，输出口。当一个摆设感觉怎么样啊？

IBOutlet: 你瞎扯啥呀？我可不是不干活。

IBAction: 啊——我可天生是个行动家，就会行动。我的工作就是出了什么事的时候拎出一个方法来。这样就把事情搞定了。你呢，不就会坐在那儿指指点点吗，净等着别人做事。

IBOutlet: 那怎么了。至少我知道事全办了。

IBAction: 没错，可是用户要干活的时候，是我帮他干的！做的保存、单击也是我干的！

IBOutlet: 听着啊，我确实就是一个实例变量而已，还得跟nib里的对象一块用，但这不是说我不重要。

IBAction: 是呀，因为编译器会忽略你，所以才重要！

IBOutlet: 是会忽略我，可是我会告诉Interface Builder很多事。你和IB的关系可不怎么样吧，是不是？

IBAction: 哈，教你个乖吧，IBAction里的“IB”两字就是Interface Builder的意思！

IBOutlet: 了不起吗？我的名字里也有“IB”俩字啊。

IBAction: 得，这点我们扯平了。可是Interface Builder只要看见我，就知道nib里有我要的消息，一有事就通知我。

IBOutlet: 嗯，听上去你这活，还挺重要的。

IBAction: 太谢谢了。您终于肯承认了。

【拼吾爱-最新编程资源的分享下载站】

IBAction:

哟呵，说来听听！

哦，我明白了。你知道吗？你有一样本事是我一直想要学的。

你扮谁像谁啊！把IBOutlet随便往一个变量名前面一杵，全都可以。我的语法就复杂多了，因为我得知道给我发送消息的是哪位啊。

我觉着也是。

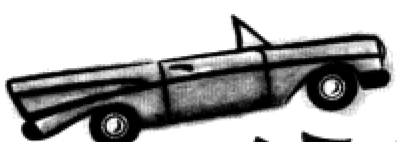
IBOutlet:

可是我和Interface Builder的关系那才叫铁呢。没有我，那代码就别想动UI里的半点东西。

行啊。一个IBOutlet变量能指向nib中的特定对象（比如一个文本字段什么的），而代码呢（嗯，很可能就是你的代码），可以利用我来改变UI，设置文本字段的内容，改变颜色等。

是什么呀？

我就喜欢自由自在！得，咱俩共事能完成工作，就挺好。

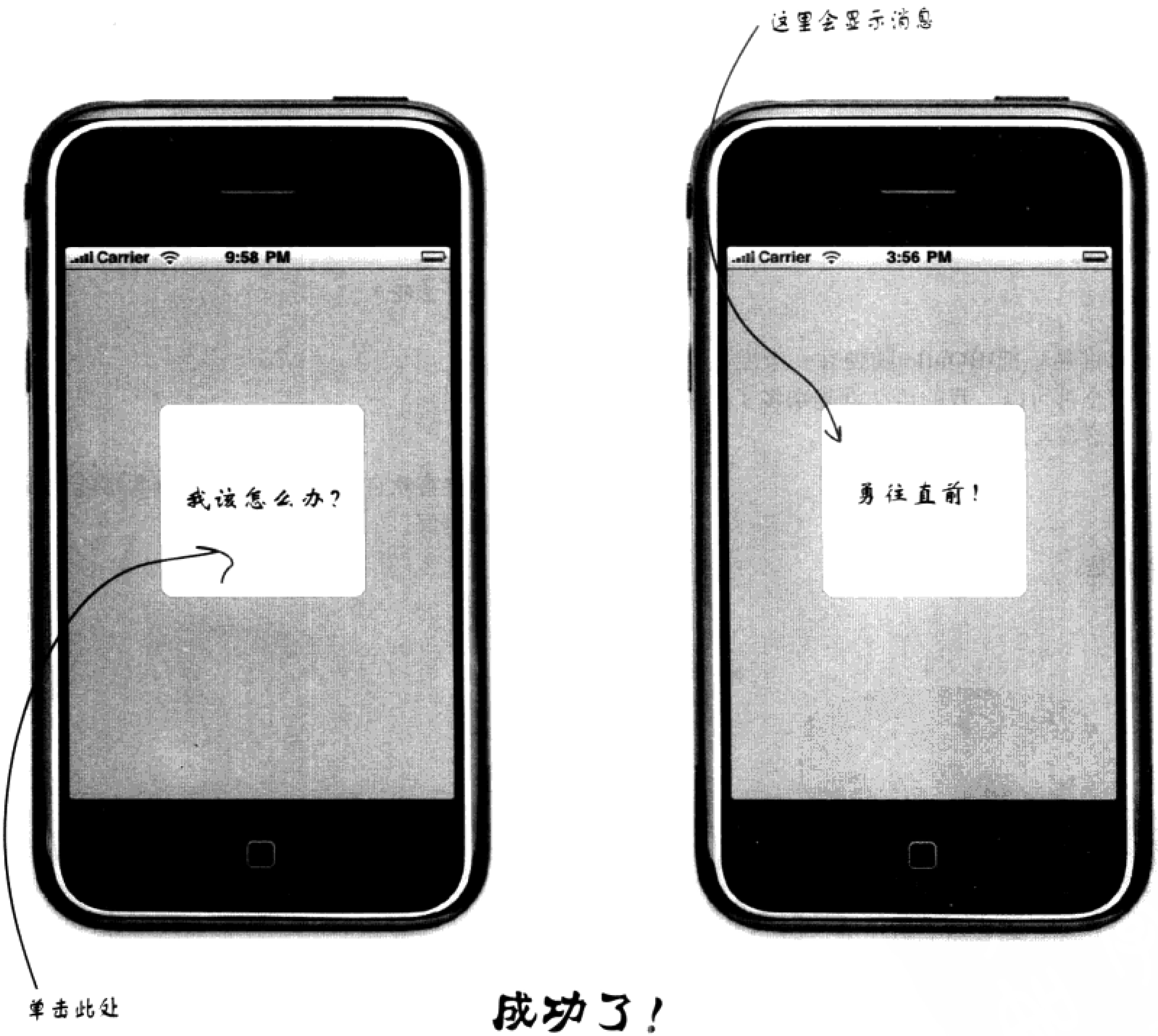


运行测试

既然一切都已经关联完毕，该上路了。请确保你在Interface Builder里的工作全都保存了，然后回到Xcode，构建并运行应用。

【拼吾爱-最新编程资源的分享下载站】

运行测试



成功了!

【拼吾爱-最新编程资源的分享下载站】





迈克至少可以做出一次决定了。

你的应用工作正常！这是各个部分合作的成果：*.xib文件描述界面，Interface Builder把它与代码关联起来，而Objective-C让它们一起正常运转。

你正在向通往App Store最卖座应用的道路上迈进。

编一个Twitter应用如何？

【拼吾爱-最新编程资源的分享下载站】

有问必答

问： 那个文件所有者（File's Owner）是什么东西？

答： Interface Builder对于什么样的类可以做nib中的文件所有者有一定要求，但默认情况下Xcode新建项目时会让主视图控制器做主视图的所有者。这也是为什么我们不用对此进行任何修改的原因。由于文件所有者被设置为我们的iDecideViewController，Interface Builder可以查看其头文件来确定其中是否存在名为descriptionText的IBOutlet和名为buttonPressed的IBAction。当你将UILabel的引用输出口和文件所有者的descriptionText关联起来时，Interface Builder会将必要的信息保存起来，这样在应用程序加载nib的时候，该引用可以在iDecideViewController中被正确设置。对于TouchUpInside事件来说，情况也基本一样，只不过并非是将组件和引用进行关联，而是对组件的事件与待调用的方法做关联。

记住——Interface Builder要求这个类来加载nib，并不等于不能用其他的类——只不过如果该类没有必要的属性和方法，可能会工作不正常。

问： "输出口"又是什么呢？

答： Interface Builder中有输出口和动作这两个概念，对此我们稍微解释一下。基本上，输出口就是一个对某个对象的引用；而动作就是一则消息（即方法），在发生情况时会被发送（即调用）。

问： 为什么新的字符串前面有一个@符号？

答： Cocoa Touch使用的字符串类叫做NSString。因为这个类太常用了，所以Objective-C内置支持创建这种类型的常量。要表示一个字符串常量是NSString，就在前面加上一个@符号。否则就会被当成C或C++中的普通的char *字符串常量。

关键要点

- Interface Builder创建nib文件（扩展名为.xib），用XML来描述GUI界面。
- Nib文件被Cocoa Touch框架所加载，并被动态转化为Cocoa Touch类的真实实例。
- 要将nib中描述的组件与你的代码关联起来，需要使用IBOutlet和IBAction标签。
- 用Xcode来维护应用程序的代码和文件。
- Xcode是项目开发的枢纽，提供各种功能，包括编辑代码、构建应用以及在运行起来后进行调试。
- iPhone模拟器令你能够在Mac电脑上测试应用程序，而不需要一台真实设备。

【拼吾爱-最新编程资源的分享下载站】

对号入座

用线将iPhone开发用到的各项事物与其正确的描述联系起来。

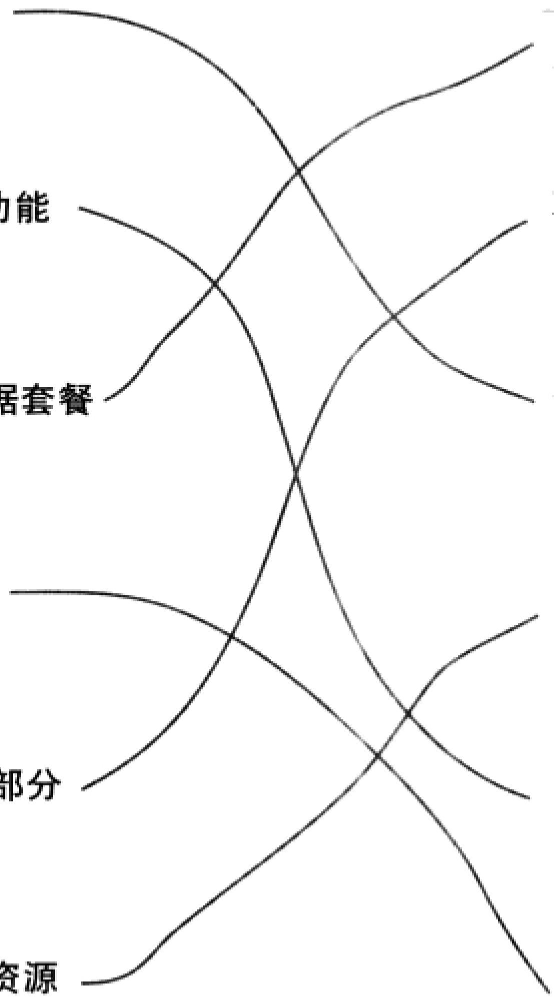
事项	描述
`IBOutlet`	一种常用于iPhone的计划套餐，和其他大多数移动电话的都不一样。
Xcode的功能	Xcode、Instruments、Interface Builder以及iPhone模拟器。
无限量数据套餐	代码中对界面的引用。
`IBAction`	图像、数据库、图标文件等。
SDK组成部分	对代码和资源进行编辑与维护，调试代码，以及为部署应用做准备工作。
应用程序资源	表示一个方法会被调用，来响应某个事件。

【拼吾爱-最新编程资源的分享下载站】

对号入座答案

用线将iPhone开发用到的各项事物与其正确的描述联系起来。

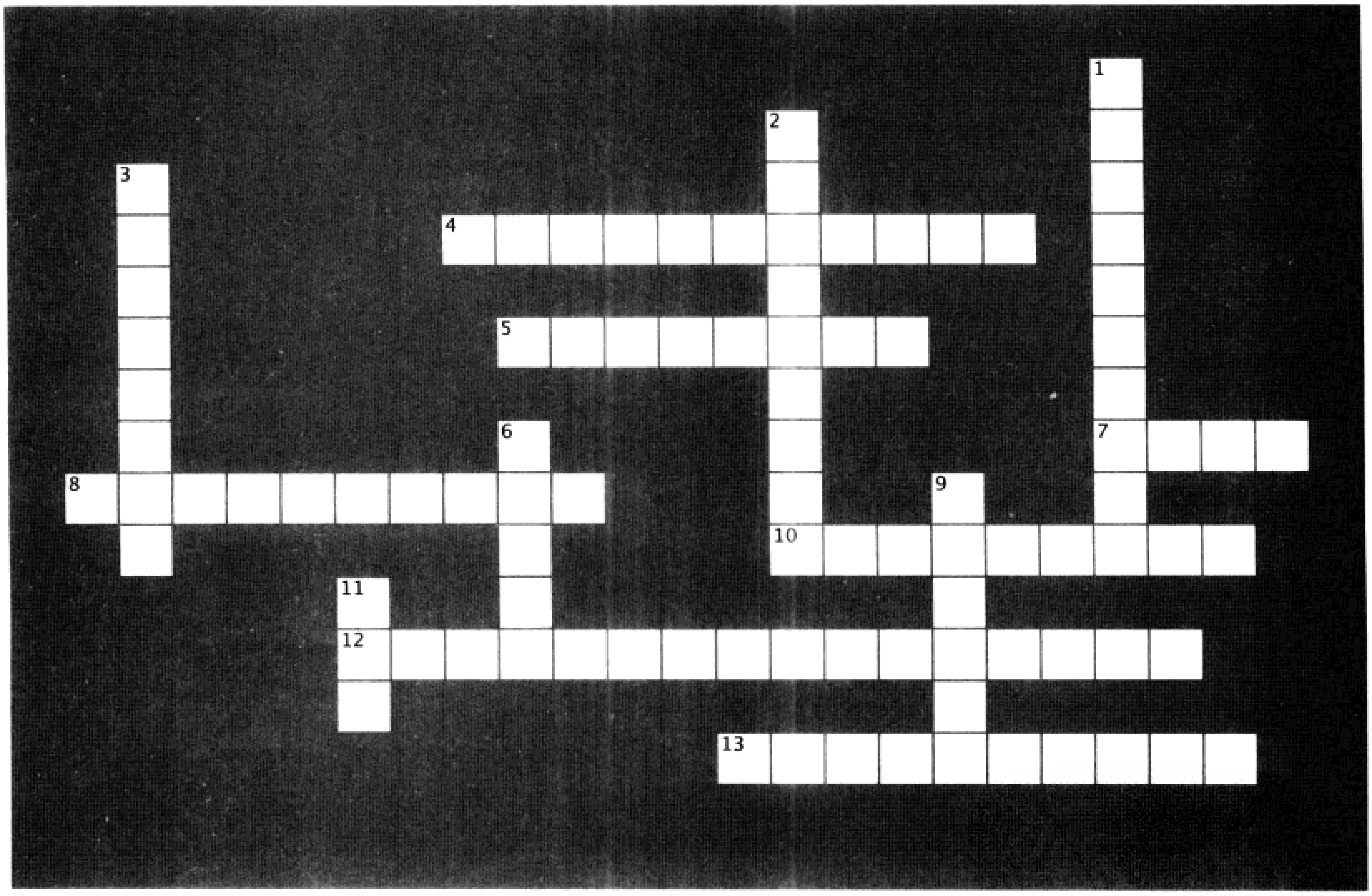
事项	描述
IBOutlet	一种常用于iPhone的计划套餐，和其他大多数移动电话的都不一样。
Xcode的功能	Xcode、Instruments、Interface Builder以及iPhone模拟器。
无限量数据套餐	代码中对界面的引用。
IBAction	图像、数据库、图标文件等。
SDK组成部分	对代码和资源进行编辑与维护，调试代码，以及为部署应用做准备工作。
应用程序资源	表示一个方法会被调用，来响应某个事件。



【拼吾爱-最新编程资源的分享下载站】

iPhone纵横字谜

开动你的脑筋，来认识一下我们在这一章中用到的新术语。



横向提示

4. 模拟器无法可靠测试的东西。
5. 用于建立一条从实现代码到视图的向外连接。
7. 描述iPhone应用中的每幅画面的术语。
8. 用来编写iPhone应用的框架。
10. 用来组织应用中的图像的文件夹。
12. 开发iPhone应用界面的IDE的名字。
13. 在Xcode当中，用来提供要用到的一些类。

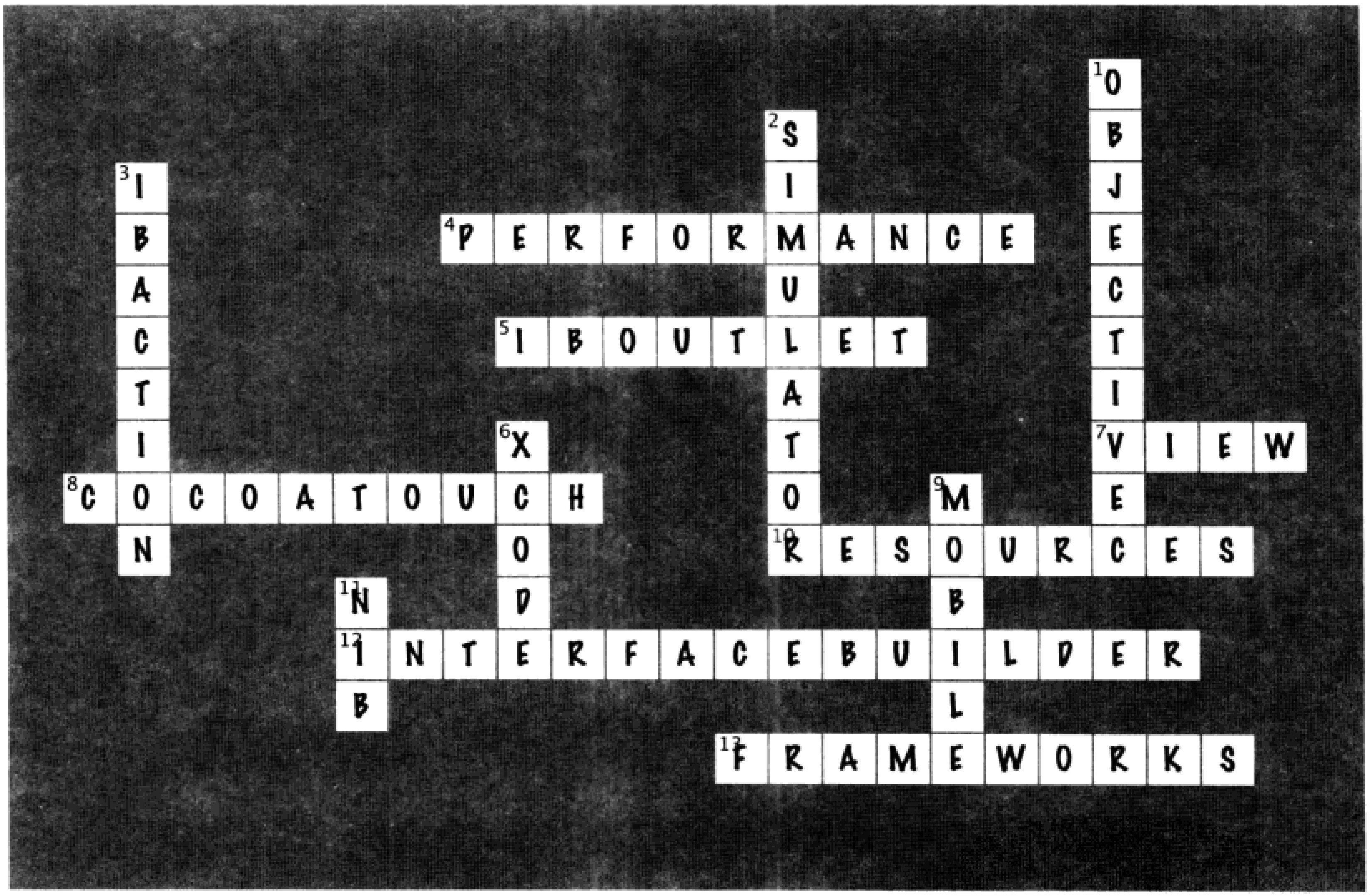
纵向提示

1. 用于编写iPhone应用的语言。
2. 可以在桌面电脑上测试应用的东西。
3. 用来在代码中接收事件，并触发一些事情。
6. 用于Objective-C的编辑器的名称。
9. iPhone所属的设备种类。
11. 用来创建视图的文件的名称。

【拼吾爱-最新编程资源的分享下载站】

iPhone纵横字谜答案

开动你的脑筋，来认识一下我们在这一章中用到的新术语。



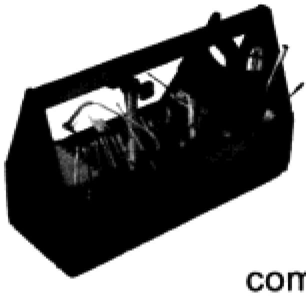
横向提示

4. 模拟器无法可靠测试的东西。[PERFORMANCE]
5. 用于建立一条从实现代码到视图的向外连接。[IBOUTLET]
7. 描述iPhone应用中的每幅画面的术语。[VIEW]
8. 用来编写iPhone应用的框架。[COCOATOUCH]
10. 用来组织应用中的图像的文件夹。[RESOURCES]
12. 开发iPhone应用界面的IDE的名字。[INTERFACEBUILDER]
13. 在Xcode当中，用来提供要用到的一些类。[FRAMEWORKS]

纵向提示

1. 用于编写iPhone应用的语言。[OBJECTIVEC]
2. 可以在桌面电脑上测试应用的东西。[SIMULATOR]
3. 用来在代码中接收事件，并触发一些事情。[IBACTION]
6. 用于Objective-C的编辑器的名称。[XCODE]
9. iPhone所属的设备种类。[MOBILE]
11. 用来创建视图的文件的名称。[NIB]

【拼吾爱-最新编程资源的分享下载站】



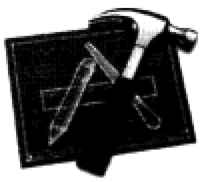
你的iPhone工具箱

你已经消化吸收了第1章，现在可以把基本的iPhone应用交互功能添加到你的工具箱中了。如果想得到一份完整的书中功能提示的列表，请访问http://www.headfirstlabs.com/iphonedev。



在Interface Builder中创建视图

视图是由nib（*.xib）文件组成的，而GUI界面用Interface Builder来编辑。



...然后代码令视图运转起来...

这里的代码几乎总是在Xcode中用Objective-C来写的。



...对于其他任何资源，全部打包到应用程序当中。

图像以及其他数据也在Xcode中统一管理，所以所需要的全部文件都能轻松处理。

【拼吾爱-最新编程资源的分享下载站】

【拼吾爱-最新编程资源的分享下载站】

2 iPhone应用的模式

你好，@twitter!



应用都具有许多活动部件。好吧，实际上它们没有真正的活动部件，但是它们确实具有很多UI控件。一个典型的iPhone应用，可不会仅仅有一个按钮，现在是时候让我们来构建一个这样的应用了。这要使用一些更复杂的部件，意味着你也需要比以前更加注意应用的设计。在这一章里，你将学到一些用在iPhone SDK里的基本的设计模式，并学会如何组建出一个更大型的应用程序。

【拼吾爱-最新编程资源的分享下载站】

Head First 休闲吧

作者提示：Head First对迈克的感情问题不承担任何责任。

迈克又回来了。他有一个很棒的女朋友名叫瑞妮，但是他们之间最近出现了一点问题。她觉得他没有充分表达自己的感觉。



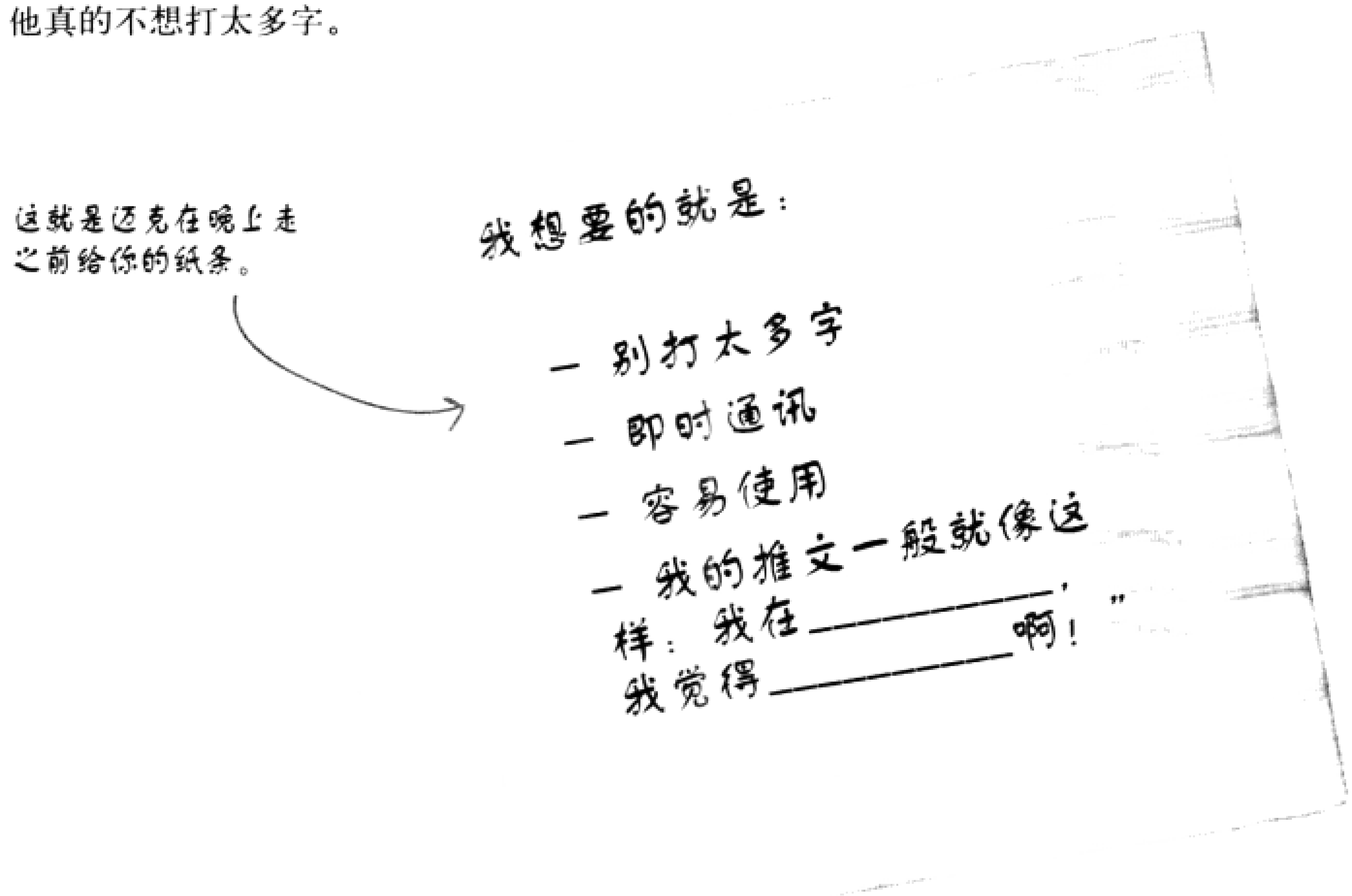
有一个（马上就会有）应用就是做这个的。

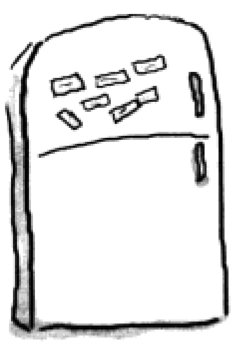
通过可靠的设计，再加上Interface Builder控件库中就自带的一些基本控件，你就可以让迈克随时上Twitter发推文。但是首先要解决一个问题，他的推文里要说点什么呢？

【拼吾爱-最新编程资源的分享下载站】

首先我们需要找出迈克(真正)的需求

迈克没什么城府。他只想要个简单易用的界面，能同Twitter对话就行，他真的不想打太多字。





应用魔力贴

现在我们知道迈克想要什么了，那我们该做些什么呢？把下面的魔力贴按照顺序排列起来，好让你跟着步骤做就可以弄出他想要的应用。

决定应用的布局

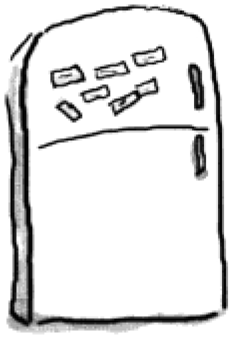
打造界面

把结果发送到Twitter

弄明白如何使用控件

处理数据

【拼吾爱-最新编程资源的分享下载站】



应用魔力贴答案

现在我们知道迈克想要什么了，那我们该做些什么呢？把下面的魔力贴按照顺序排列起来，好让你跟着步骤做就可以弄出他想要的应用。

决定应用的布局

在开始任何编码之前，先画出你的构思草图。

打造界面

有些人先写后端代码-我们则根据项目的不同来回变，不过现在你刚入门，我们还是先做界面部分吧。

弄明白如何使用控件

当你开始着手一般的应用设计之后，需要查看一下文档，弄清楚怎么用你选择的控件实现意图。

处理数据

在这里我们需要管理来自控件的数据……

把结果发送到Twitter

这最后一步我们也会帮你完成。

有问必答

问： 该如何设计应用布局？

答： 在刚开始入门时，我们会让你做一些二选一，但通常来说，很重要的一件事就是好好考虑你的应用该做些什么，首先要专注于这些功能。

问： 我们总是会以画草图开始吗？

答： 是的！良好的软件设计一开始总是要弄清目标是什么东西，而用户又将如何使用它。给迈克的应用是用在Twitter上的，而他需要的功能是从他自己的感受和想法中选出一些发上去。就是这些！

问： 如何与Twitter对话呢？

答： 别担心，我们会给你一些代码，帮你完成这部分。

不过，这仅供参考，Twitter的API文档真的很不错。我们会给你必要的代码，不过想要加上些功能的话，也完全随你！

问： 每个控件的工作方式都与其他的控件不一样吗？

答： 对于大部分控件来说，并非如此——只要你学会了几种基本的模式，就可以自己摸索出SDK中大多数东西该怎样使用。某些控件偶尔会有一些特殊的地方，不过大体来看，你会对它们逐渐变得熟悉起来。

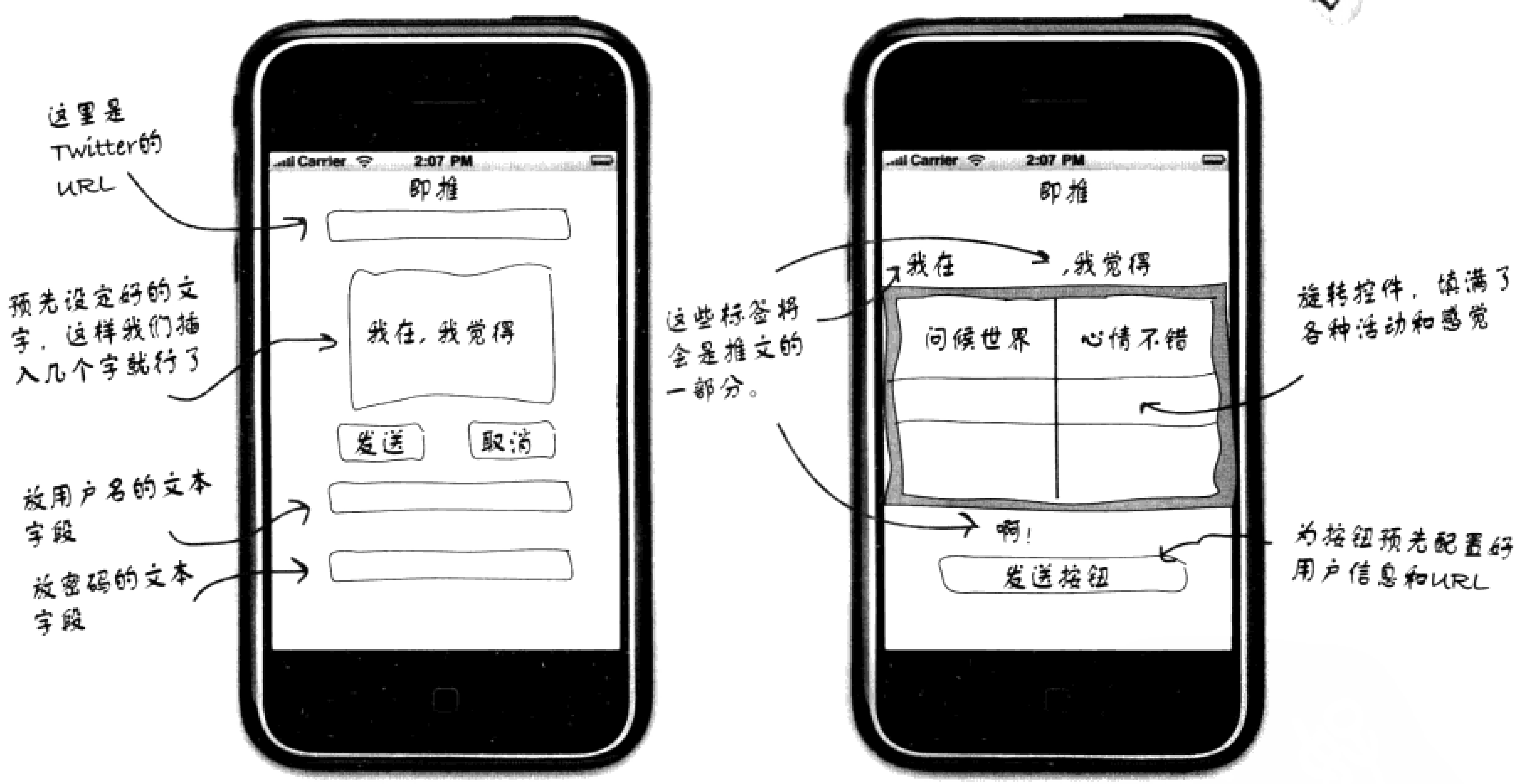
【拼吾爱-最新编程资源的分享下载站】

搭建应用的布局



下面是要评估的两种设计方案。请你基于美观、易用以及符合iPhone应用行为标准等原则，为迈克选出一个更合适的。

方案1　　　　　　　　方案2



哪个应用更合适？ ....

为什么？（详细点） ....

....

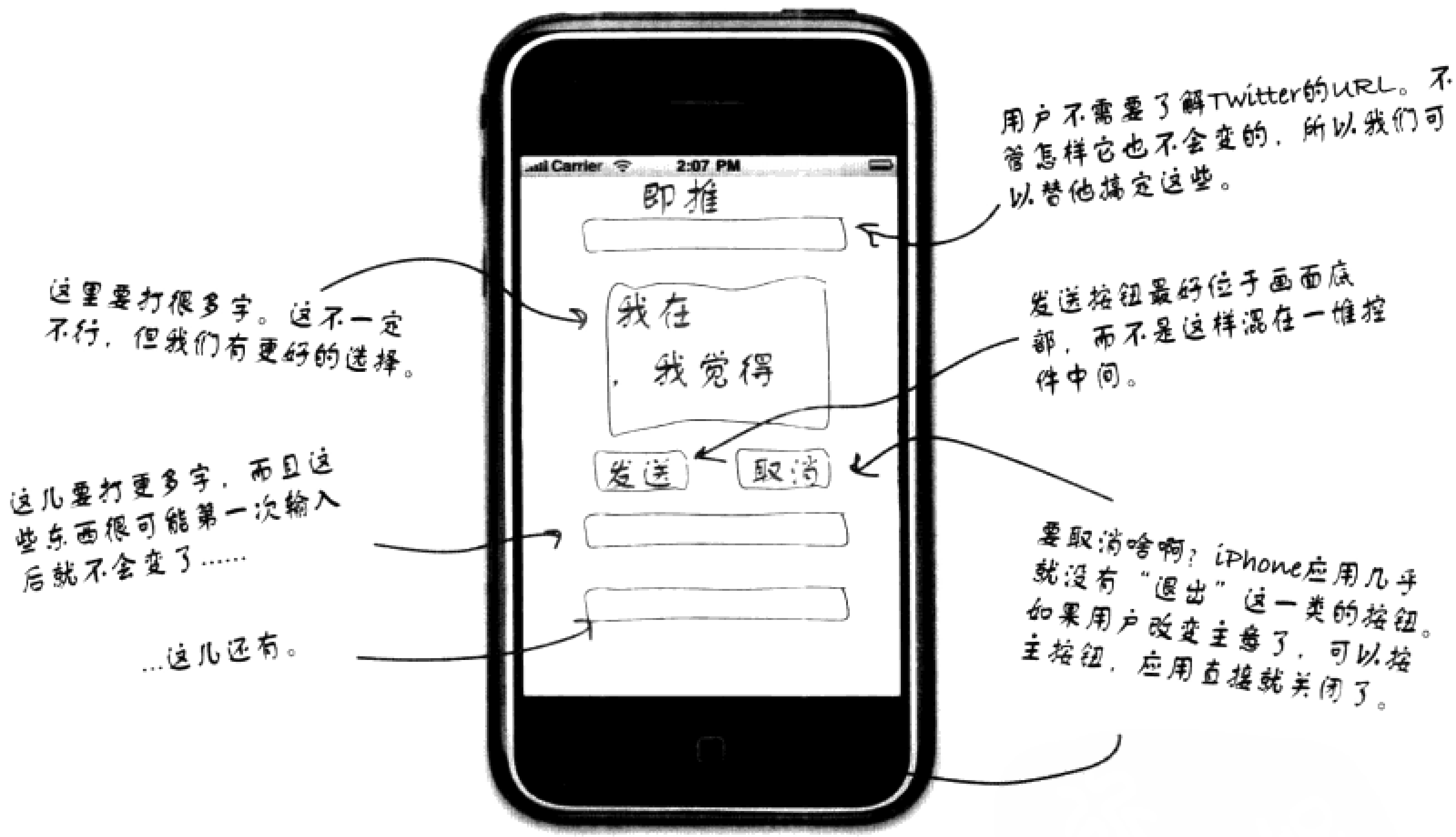
为什么不选另一个？ ....

【拼吾爱-最新编程资源的分享下载站】

搭建应用的布局

下面是要评估的两种设计方案。请你基于美观、易用以及符合iPhone应用行为标准等原则，为迈克选出一个更合适的。

方案1

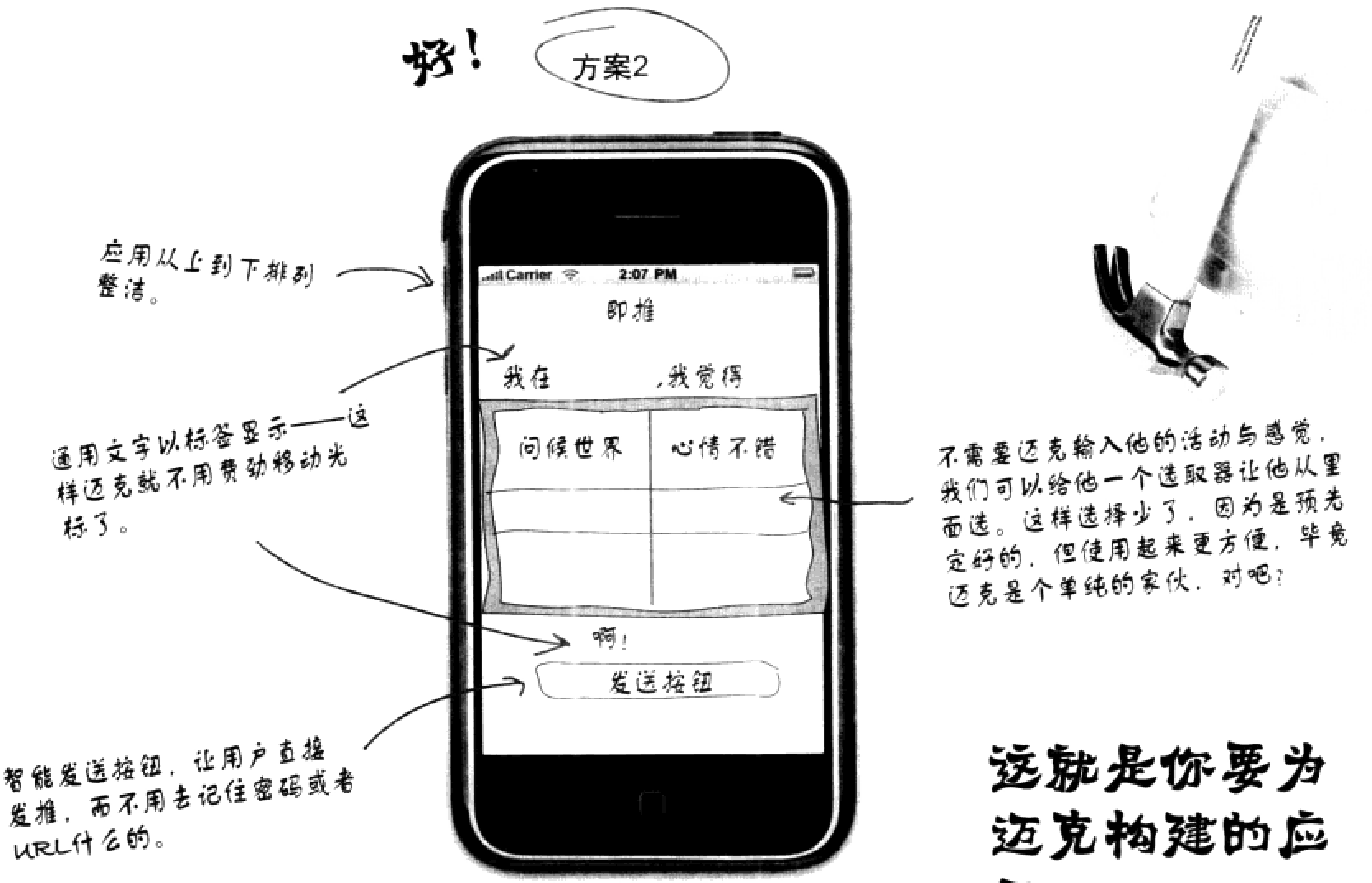


哪个应用更合适？ 2号方案。

为什么？(详细点) 方案2可以少打很多字，而且既然用户不会经常改用户名和密码，所以没理由把这些也放在主视图里，让用户每次运行这个应用都得输入。

为什么不选另一个？方案1要打很多字，要记住很多设置。按钮布置也让人犯晕。

【拼吾爱-最新编程资源的分享下载站】



这就是你要为迈克构建的应用。

有问必答

问： 我真的需要那么在意易用性和美观吗？

答： 正是美观易用造就了iPhone的成功，所以苹果公司会誓死捍卫这一点的。更重要的是，不得到他们的同意，你别想把任何东西放到App Store或者别人的iPhone上去。苹果公司已经卖出了超过10亿份应用——如果你的应用不符合iPhone的外观感受，或者难以使用，人们就会去用其他某人的应用而对你弃之不顾。

问： 我们去掉了用户名、密码和URL的字段。我能理解对URL这么做，但剩下这两个是怎么回事？

答： 如果你的应用需要一些配置信息，而用户又不会经常修改它们，就应该将其摒弃出主工作流程之外。苹果公司甚至为此专门提供了一块地方，叫做设置束（Settings bundle），会和标准的iPhone设置放在一块儿。在本章我们不打算使用它（我们直接把这些值写在程序里），不过以后我们会展示怎么把东西放进设置页面里去。登录信息这样的东西放那里正合适。

问： 我怎么知道什么样的设计苹果公司会觉得好或者觉得美观呢？

答： 呵呵，你该问……继续吧，翻下一页。

【拼吾爱-最新编程资源的分享下载站】

应用之中，设计为王—iPhone人机界面指南

iPhone的人机界面指南（HIG）是苹果公司发行的一份文档，用来指导开发在App Store中出售的iPhone应用。你可以从http://developer.apple.com/iphone下载这份文档。这可不是他们助人为乐做好事，一旦你提交了应用去审查，就相当于你同意了应用要遵从这个HIG。

我们一定要强调这一点：你必须遵循HIG，因为苹果公司的审核过程是很详细的，如果你不符合规定，他们会拒绝你的应用程序。随之而来的就是抱怨，在博客上愤怒地发文，然后照办。这个话题我们就说到这里。

苹果公司还发布了另外几个指南和教程，包括iPhone应用程序开发指南。这是另一个信息来源，解释了如何处理不同设备之间的差异，比如iPhone和iPod Touch。不注意应用在iPod Touch上的表现，是另一种让你的应用被App Store拒掉的方法。

注意：尽管作者不建议去测试这些被App Store拒绝的方法，但我们负责任地讲，它们都很管用。

应用类型

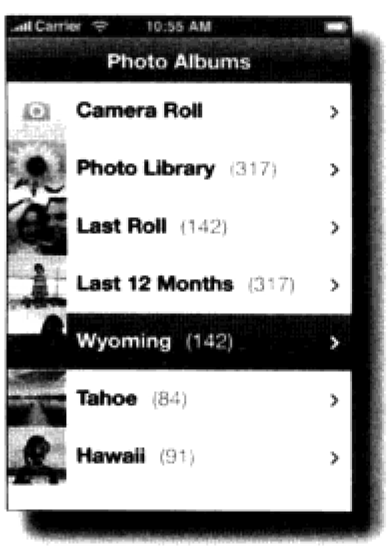
HIG详细总结了常见的为iPhone开发的应用程序，归纳为3种主要类型。每种类型的应用都有不同的使用目的，因此提供了不同的用户体验。在开始做GUI之前，首先要想好你做的是什么类型的应用程序，这有助于顺利得到良好的界面设计。

沉浸式应用



游戏是经典的例子，沉浸式应用就像这个模拟水平仪，需要高度定制的界面，好让用户能与设备互动。所以，HIG指导意见在这种情况下不是很关键。

生产力应用



帮助用户管理信息并完成任务。信息是分层的，可以导航进入下层，从而获取更详细的内容。

工具类应用



为用户获取特定信息，尽可能地简化交互或者配置设置。

通常比生产力应用的界面设计复杂，被认为需要与HIG保持高度一致。

【拼吾爱-最新编程资源的分享下载站】

对号入座

下面是一组不同的应用程序创意。对于每个创意，请思考它到底属于什么类型的应用，将其与右边的应用类型对应起来。

应用描述	应用的类型
InstaTwit 1.0：让你可以用最少的输入发推。	
新闻阅读器：展示一系列新闻分类，你可以选中一些报道，查看详细内容。	沉浸式应用
弹珠游戏：一个滚玻璃球的游戏，用加速计作为控制手段。	工具类应用
秒表工具：通过触摸屏幕来启动或停止的秒表。	生产力应用
发票管理：列出餐费，让你可以深入下层，查看每张发票。	

【拼吾爱-最新编程资源的分享下载站】

对号入座答案

将每个应用的描述与它的应用类型对应起来。

应用描述	应用的类型
InstaTwit 1.0：让你可以用最少的输入发推。	
新闻阅读器：展示一系列新闻分类，你可以选中一些报道，查看详细内容。	沉浸式应用
弹珠游戏：一个滚玻璃球的游戏，用加速计作为控制手段。	工具类应用
秒表工具：通过触摸屏幕来启动或停止的秒表。	生产力应用
发票管理：列出餐费，让你可以深入下层，查看每张发票。	

由于只有一幅画面又不用打字，即推更像一个工具应用。

由于这个应用是列表驱动、逐级细化的界面，所以是生产力类的

用加速计控制，滚动大个玻璃球…

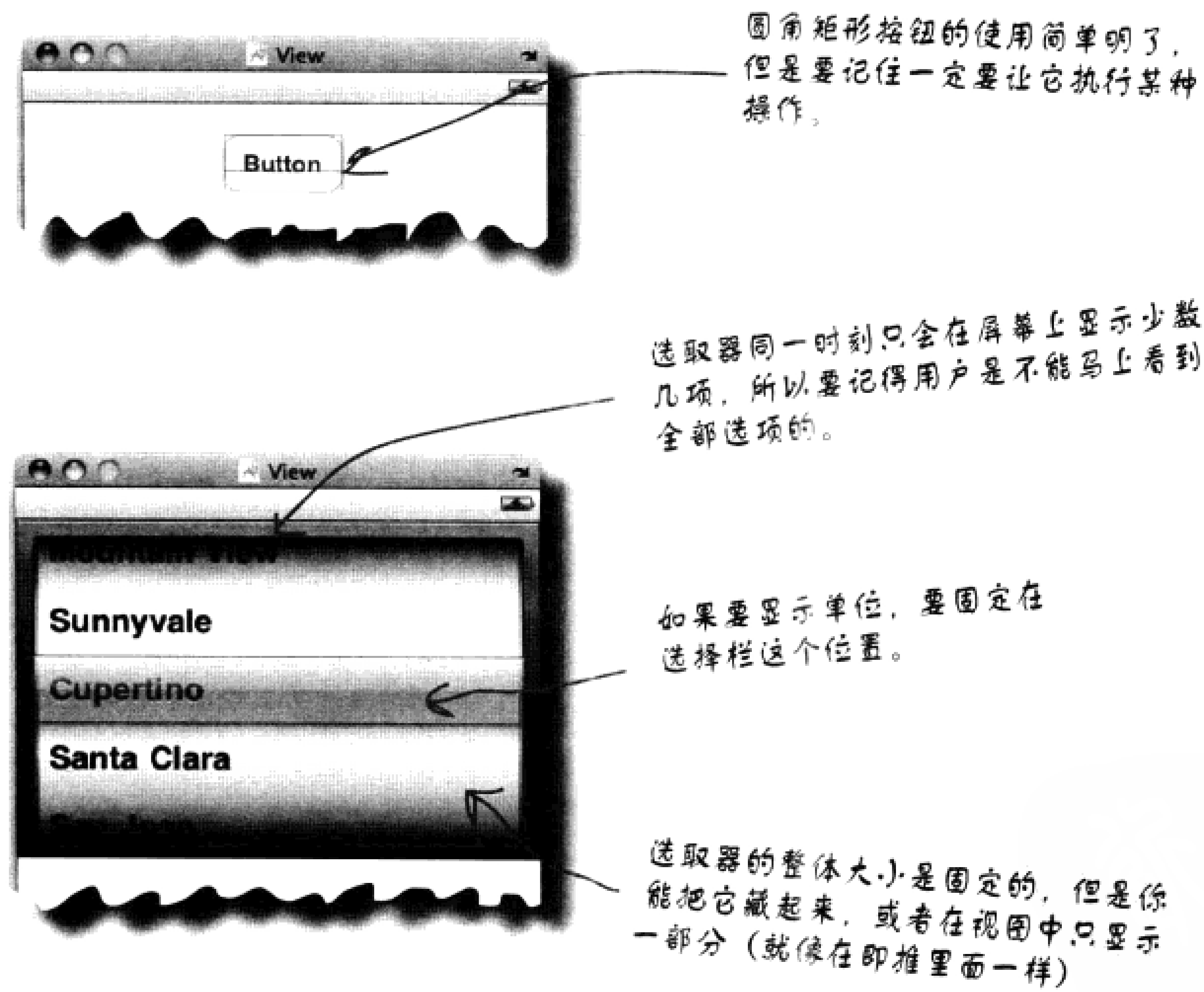
我们需要专门的秒表界面，不用实际处理数据

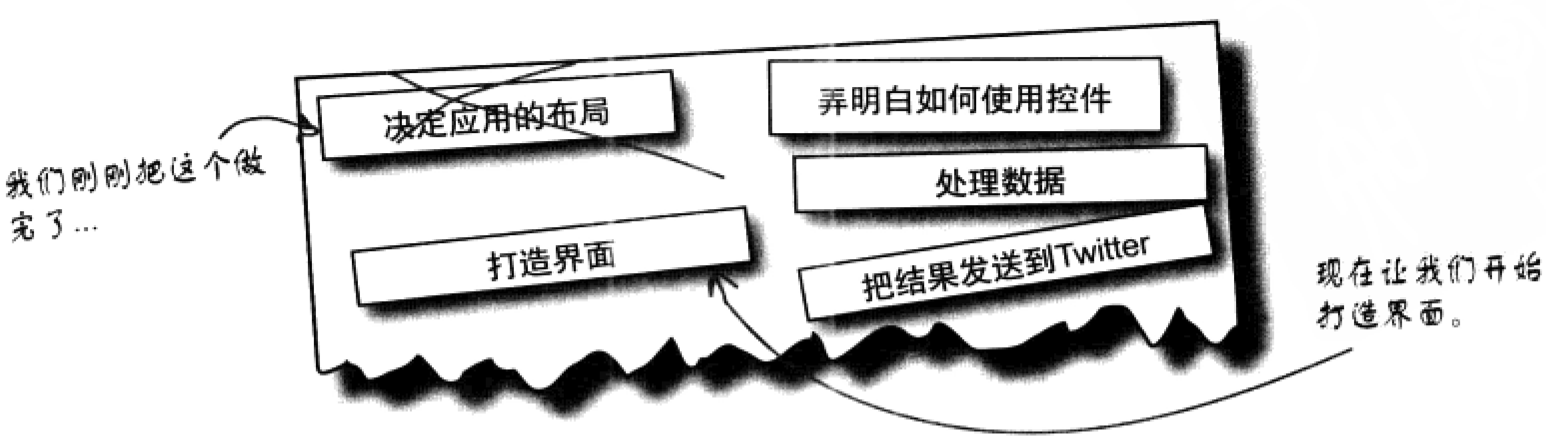
要处理大量数据：表格、逐级细化的发票——绝对是生产力类的

【拼吾爱-最新编程资源的分享下载站】

HIG关于选取器与按钮的指导意见

HIG中有一节专门讲如何正确使用标准控件，其中就包括了我们为即推选中的两种。在用控件打造视图之前，最好先大致看一下苹果公司的推荐使用方式。你会在HIG的第9章《应用程序控件》中找到这些信息。

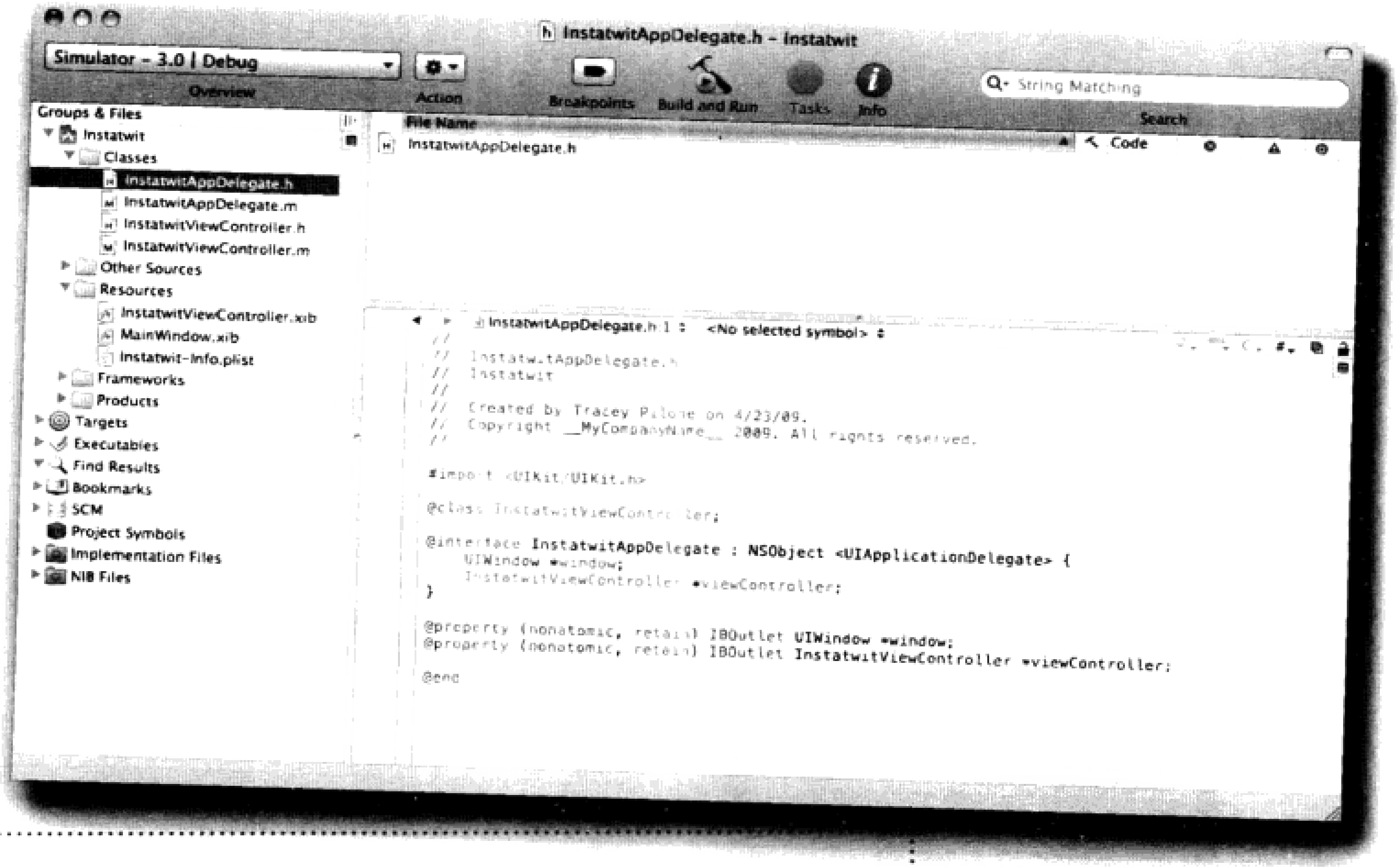




【拼吾爱-最新编程资源的分享下载站】

为InstaTwit（即推）创建一个基于视图的项目

打开Xcode之后，选择File → New Project。InstaTwit和iDecide一样，只有一幅画面，我们也不打算翻转它或者做其他花哨的东西，所以还是选择View-based Application（基于视图的应用程序）这一项，并将项目命名为Instatwit。





新建项目的类型不必和你的应用类型相同。

比如，一个生产力应用有多种选择，可以做成基于视图的应用程序、基于窗口的应用程序、基于导航的应用程序或者是标签栏应用程序。

本书后面会使用其他的项目类型。

着手布局视图

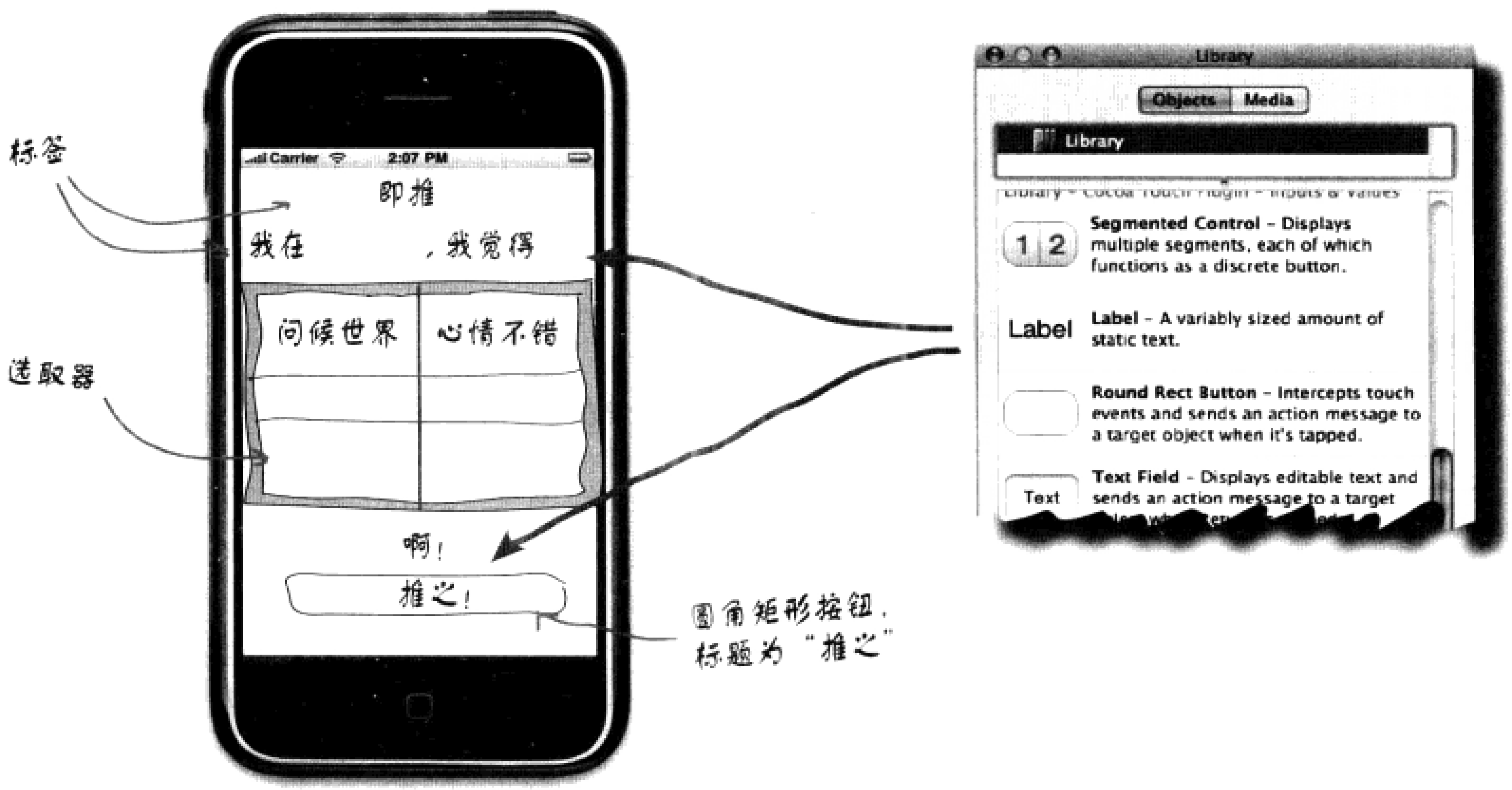
既然我们已经有生成的代码了，那就从界面开始吧。修改界面要编辑nib(.xib)文件。双击资源文件夹中的InstatwitViewController.xib，会启动Interface Builder。

【拼吾爱-最新编程资源的分享下载站】

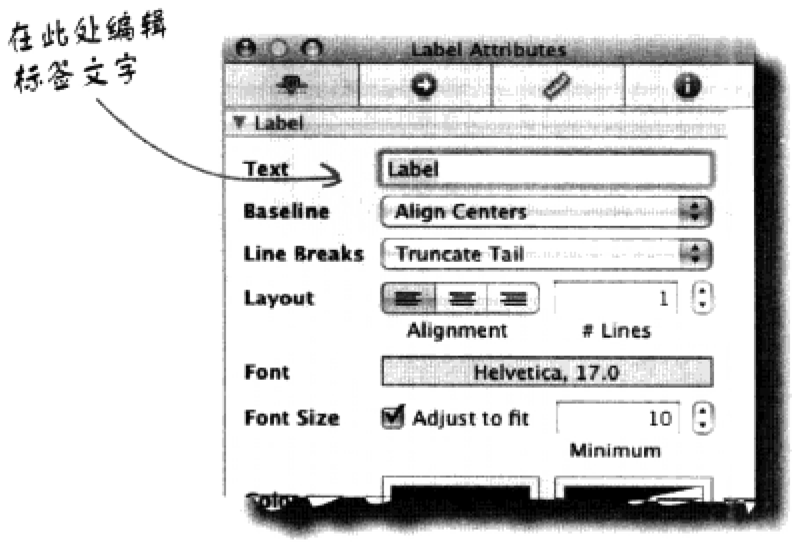
练习

现在是打造视图的时间。使用拖放方法，将元素按照需要，从Interface Builder库中拉到视图中。

1 在库中找出每个元素（我们已经给出了正确名称），将它们拖放到视图窗口中。



2 选择最顶上的标签并按下⌘1。就会启动查看器（Inspector）。



3 编辑标签与按钮的标题文字，改为“我在”、“，我觉得”和“啊！”先不要担心选取器的值。

保存以后，你的视图看起来就会像这样……

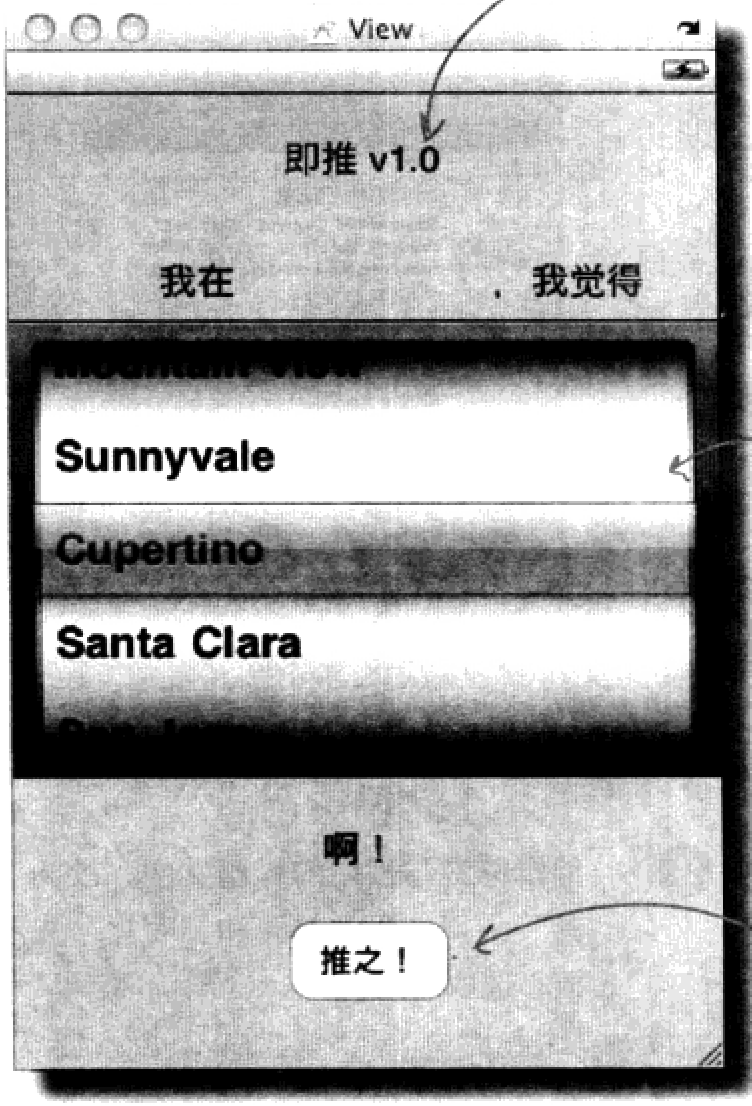
【拼吾爱-最新编程资源的分享下载站】



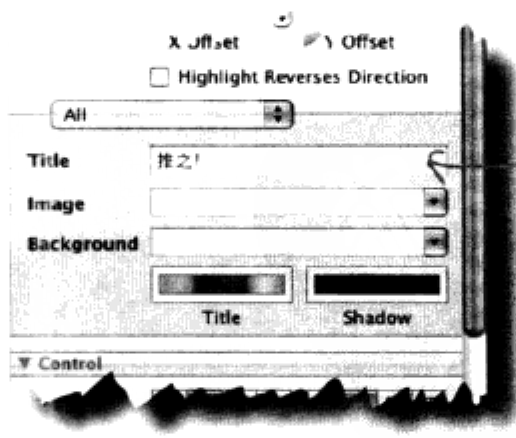
练习答案

视图打造完毕，可以使用了。这就是现在屏幕上应该看到的样子。当你将一切都调整得顺眼了之后，我们就可以运行InstaTwit（即推）了。

你的标签可能没这么大。默认情况下，标签不会随着字体缩放，而是适应空间的大小。要想让它变大的话，只要点住标签控件边缘上的点就可以改变它的大小了。



给选取器设置值需要编写一些代码，我们马上就会着手进行。在这里你看到的是默认值。



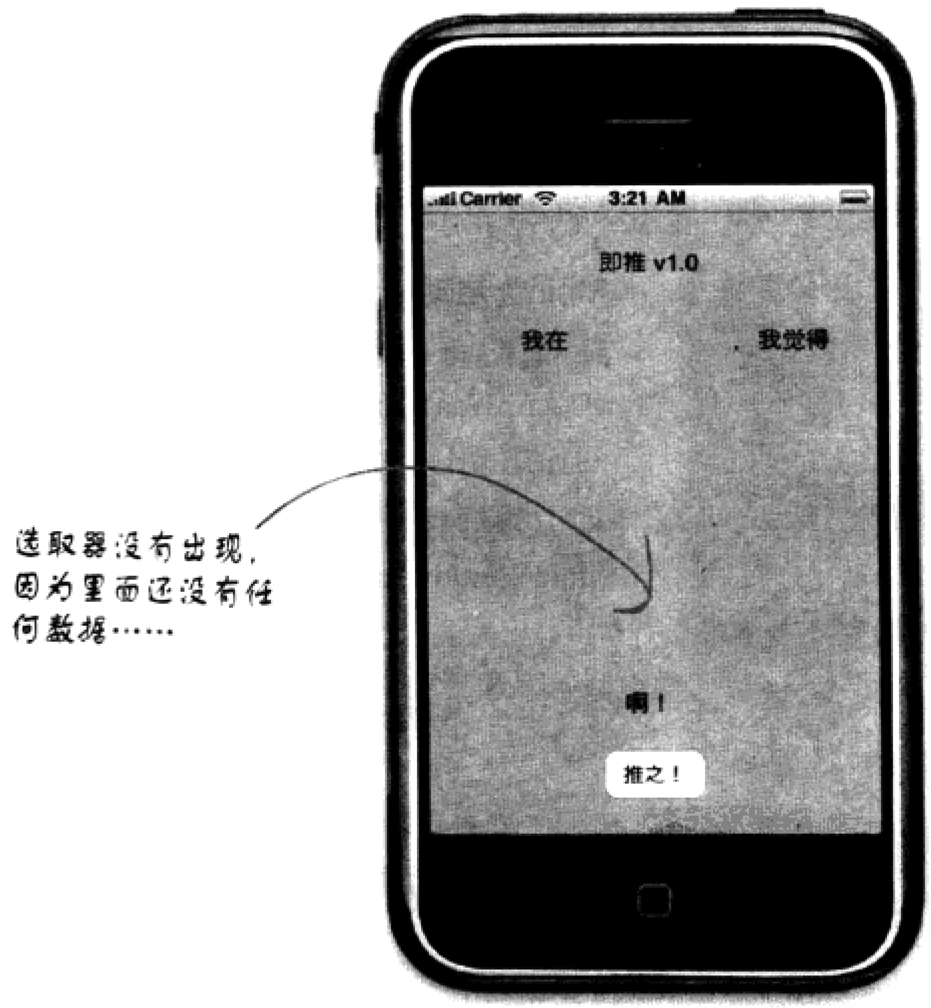
按钮的查看器略有不同——标题在窗口中更靠下的位置。

不知你注意到没有，界面仿真窗口中有一些蓝色的定位线？这些线会在你排布元素时出现在视图中，帮助你将控件居中放置，好让它们排列整齐。

【拼吾爱-最新编程资源的分享下载站】

运行测试

现在可以在模拟器中试试即推了。在Interface Builder中保存，回到Xcode，然后在Build菜单中选择Build and Debug（或者按⌘+return）。



脑力练习

要让选取器出现，需要在里面填上数据。你认为填写数据的代码应该放在哪儿？

【拼吾爱-最新编程资源的分享下载站】

根视图的生命历程

在第1章中，我们接触到了用Interface Builder来创建对视图的XML描述，描述文件叫nib，而Cocoa Touch框架会在你的应用程序中把它变成一个真正的视图。现在你也算做过一两个应用了，让我们来仔细看看幕后发生的实际情况吧。

1 像大多数其他语言一样，首先会调用main(...)。

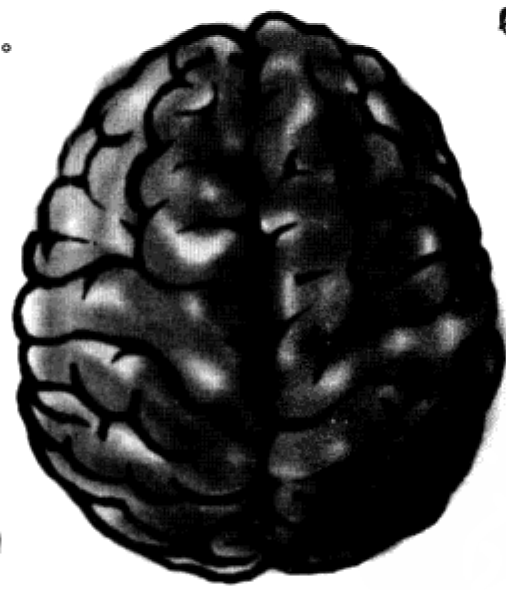
当用户启动你的应用程序时，iPhone会显示一个短暂的过场动画，将应用逐渐放大显示到屏幕上（实际显示的是一个你放在应用里面的PNG文件），然后会调用你的main方法。Main是由模板提供的，你几乎不太可能需要去改动它。



2 Main会启动Cocoa Touch应用程序。

标准的main（...）会启动Cocoa Touch的UIApplicationMain函数，这个函数会根据应用程序的Info.plist文件里面的信息，找出要加载的nib文件。对于我们使用的视图程序模板来说，就是一个名为MainWindow.xib的nib文件。

这是视图控制器。它派生自UIViewController。



3 MainWindow.xib中包含有应用程序所需的联系信息。

如果你查看MainWindow.xib，就会见到里面有一个InstaTwitAppDelegate的实例，作为它的UIApplicationDelegate，还有一个InstaTwitViewController的实例。当Cocoa框架加载这个nib的时候，将会创建一个InstaTwitViewController的实例，并让它加载我们的InstaTwitViewController.xib。

很快我们就也会好好说一说代理了。

根据 MainWindow.xib 实例化出来的InstaTwitViewController

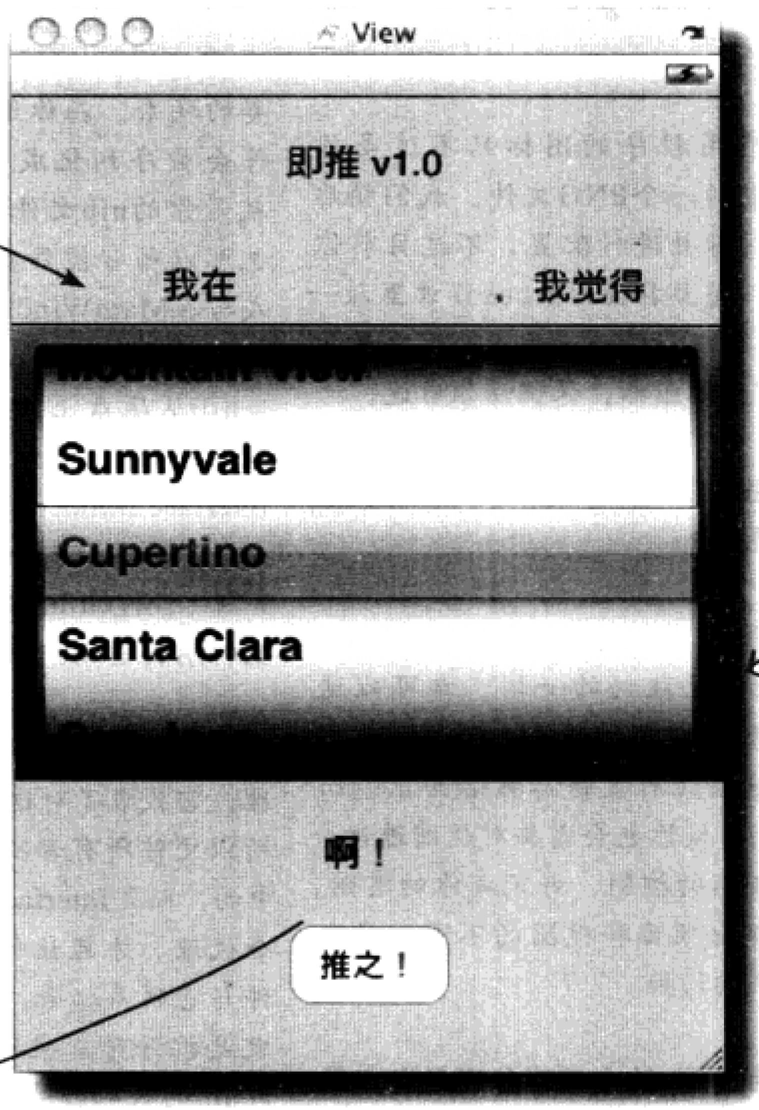
当我们构建nib时，使用了通用的文件所有者作为输出口和操作的联系对象。当这个nib被实际加载时，就会有一个真实的对象来接收这些联系。对于我们来说，就是InstaTwitViewController。

【拼吾爱-最新编程资源的分享下载站】

4 Cocoa Touch框架会根据InstaTwitViewController.xib创建我们自定义的视图。

在之前搭建nib时，我们用文件所有者这个代理对象来代表拥有nib内容的对象。现在框架替我们的InstaTwitViewController类加载nib，所以会用到一个实例来进行联系。当框架为我们的部件创建实例时，这些实例就与InstaTwitViewController的实例被联系起来了。

这是我们的视图



Nib文件中含有序列化之后的对象实例，这些对象会保持我们配置它们时的样子。通常是一些控件对象，如按钮或标签之类，不过也可以是任何可以序列化的对象。

5 **当组件上发生了事件时，就会调用我们的控制器实例中的方法。**

我们在nib中对控件与文件所有者的动作方法之间建立的联系，会被转换成控件和控制器实例之间的联系。这样一来，当控件发送出一个事件时，框架会调用我们的InstaTwitViewController实例中的方法。

现在让我们把这些知识付诸实践，并在选取器中加上一些数据。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：关于如何判断设计的好与坏，是不是有点主观了？

答： 是也不是。显然，对于什么样的UI好看，不同人有不同的看法。然而，苹果公司对于各种控件该如何使用、该怎样遵循最佳实践，都有专门的详细指南。总的来说，如果你使用了通常的iPhone控件，最好确保你的使用方式与现有的应用程序保持一致。

问：我怎么才能在我的iPhone上运行这些应用呢？

答： 要把自己写的应用安装到自己的iPhone上，你需要在http://developer.apple.com/iphone/注册，加入标准开发者或者企业开发者计划。本书中的所有内容都被设计成只用模拟器就可以运行，所以目前还用不着你去进行注册。本书后面会更详细地讲解如何将应用放到真实iPhone上的内容。

问：InstaTwit的图标太难看了。我该怎么办？

答： 应用程序的图标只不过是你的项目中的一个PNG文件。我们稍后会加入图标并进行配置，不过目前你只需要知道要往resources目录里放一个.png文件就行了——后面我们会展示一些非常酷的图标，保证让你着迷。

问：我一定得用Interface Builder来制作视图吗？

答： 并非如此。你在Interface Builder中能做到的一切，都可以通过代码来完成。Interface Builder只不过是令这些工作变得非常容易上手而已，有些时候你也会需要对视图进行一些代码级别的控制，好完成你的意图。我们会根据项目和视图的不同，在二者之间来回切换。

问：我对nib这个玩意儿还是有点晕乎。里面装的到底是我们的界面还是一般的对象啊？

答： 两种都可以放。在你用Interface Builder组装视图时，它会跟踪你使用的每个控件以及控件与其他类的联系。在你进行保存时，这些控件会被序列化成一个XML文档，这就是你的nib文件。Interface Builder也可以对非控件类进行序列化，其方式与在MainWindow.xib中存储我们的InstaTwitViewController的方式一样。当nib从磁盘上复原时，nib中的对象就会被重新实例化，并被赋予你在Interface Builder中保存下来的值。

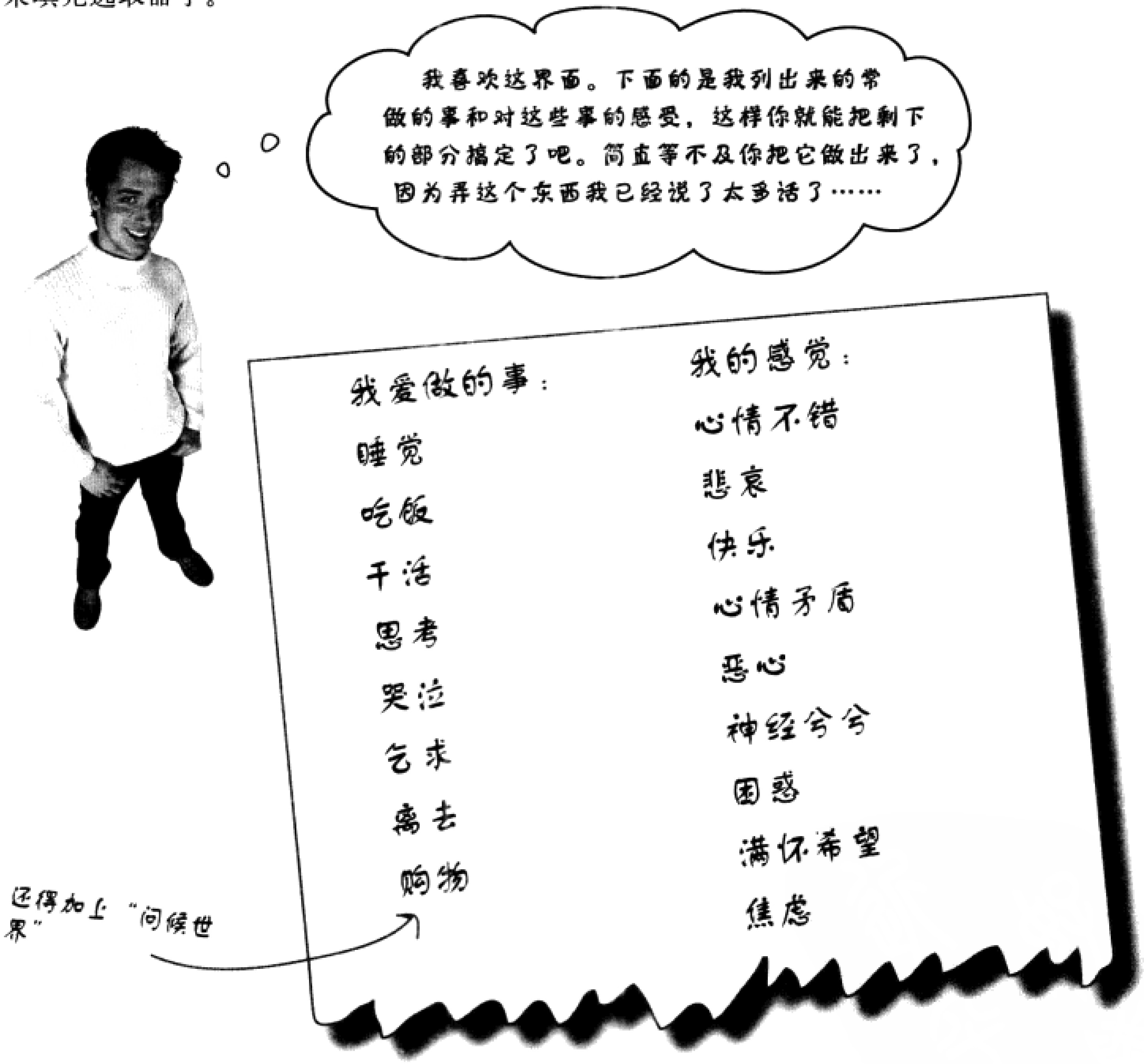
问：所以Interface Builder也会保存文件所有者吗？

答： 不会，文件所有者只是一个代理。它代表了对这个nib进行加载的类。所以文件所有者实际上不是存储在nib中的，而是Interface Builder需要这么一个代理，才能让你将视图中使用的控件与它联系起来。当nib被恢复时（也就是控件对象被实例化的时候），加载nib的代码就会将它联系到真正的所有者对象上，亦即要求加载nib的对象。

【拼吾爱-最新编程资源的分享下载站】

首先，从迈克那里获取数据

迈克觉得你整出来的UI挺不错，所以现在我们得要多一点信息，来填充选取器了。



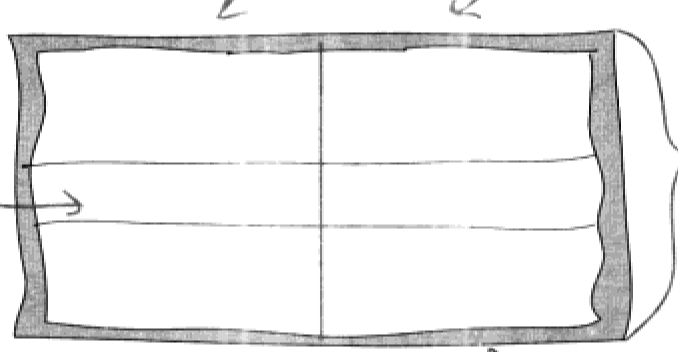
脑力练习

这些数据将被用作选取器的一部分，不过如何实现这一点呢？

【拼吾爱-最新编程资源的分享下载站】

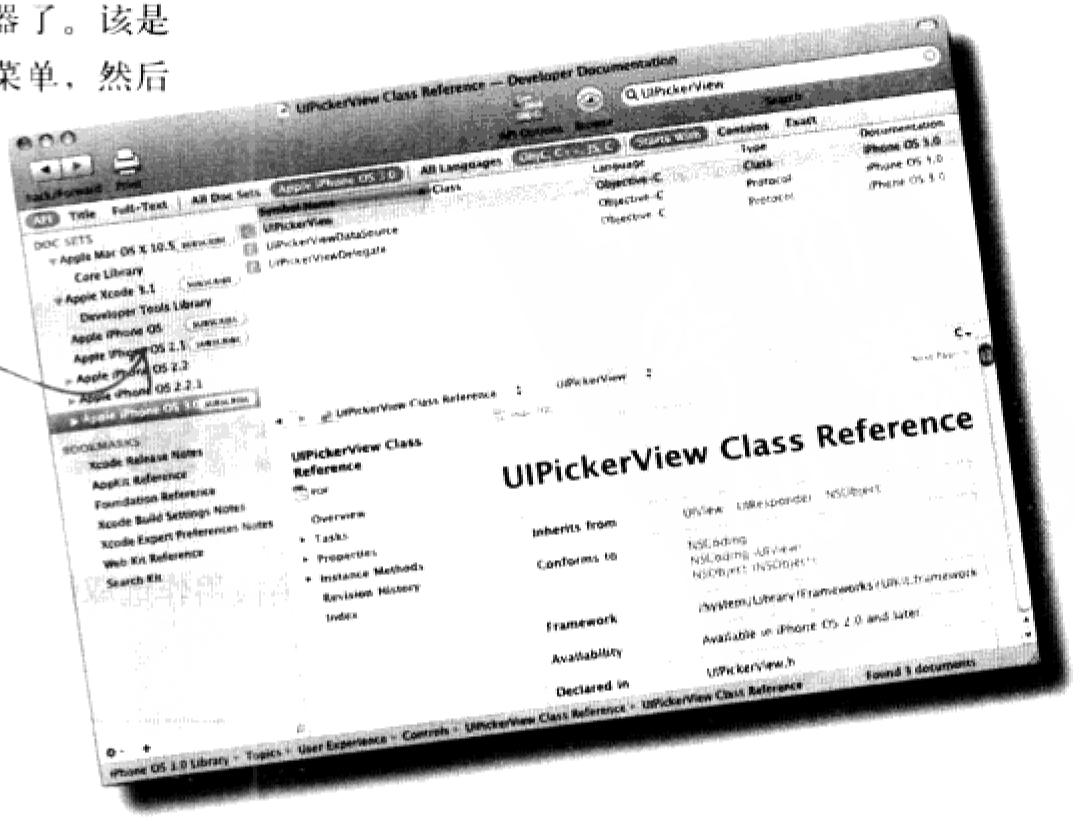
当你需要受控的输入时，就用选取器

对于我们的情况来说，在应用中采用选取器是一个理想的选择。根本不用打字，还能让迈克选中一些东西作为输入。在你向选取器加入数据之前，需要先了解一些术语。



不能确定的时候，就查查苹果公司的API文档

到了现在，你应该已经开始想着怎么实现选取器了。该是拿起API文档的时候了。在Xcode里，找到Help菜单，然后选择Documentation选项。

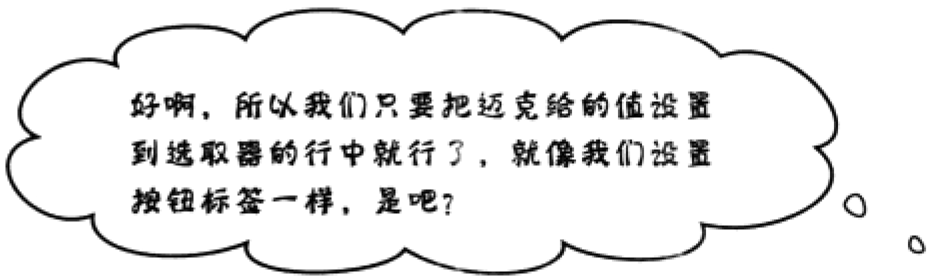


搜索“UIPickerView”，就会出现关于这个类的全部信息，靠这个类你就可实现选取器了。

【拼吾爱-最新编程资源的分享下载站】

将迈克的数据填入选取器的各行条目中

选取器需要了解自己有多少行和多少列。这些信息是由迈克提供的句子决定的。





选取器是不一样的。

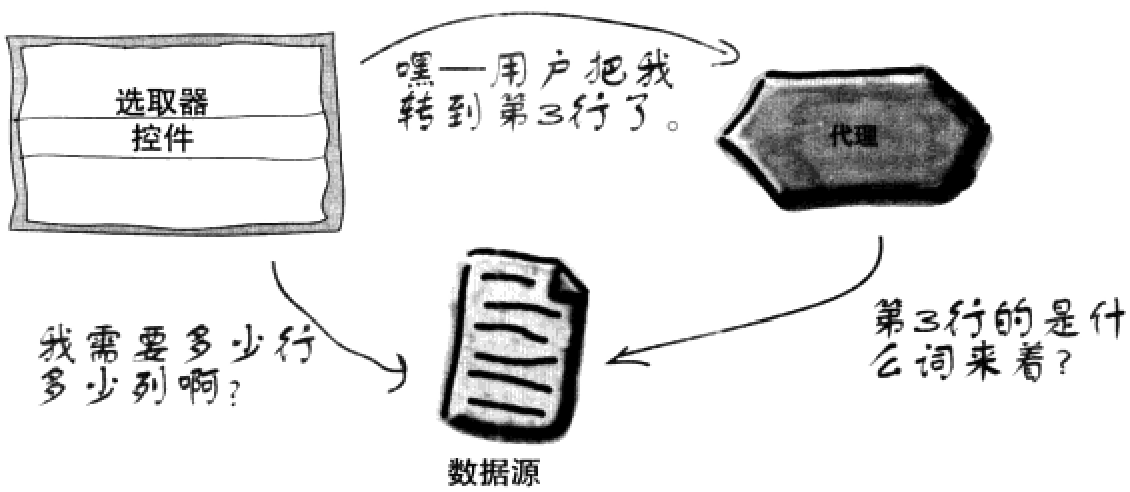
选取器不想被动地被告知要做什么事，而是在需要数据时主动请求你的输入。在那些可以使用大量数据的控件身上，你将会经常见到这种模式，比如选取器以及后面将会出现的表格视图。下面让我们更仔细地看一看……

【拼吾爱-最新编程资源的分享下载站】

选取器从数据源获取它们的数据……

Cocoa Touch框架中的大多数元素，都有数据源与代理的概念。每个UI控件都负责控制屏幕上的显示（酷酷的旋转指示，用户拨动转轮时的动画，诸如此类），但却一点也不知道自己将要显示什么数据，或者在某些东西被选中时要做什么么事。

数据源架起一座桥梁，连接了控件和它需要显示的数据。控件会向数据源请求所需的数据，而数据源负责提供信息，信息的格式也要符合控件的要求。对于我们这个应用来说，数据源要给选取器提供组件（或者说列）的数目以及总行数。不同的控件需要的数据源种类也不同。对于选取器来说，需要的是**UIPickerViewDatasource**。



...而在发生情况时则会告诉它们的代理。

代理负责元素的行为。当某人选中了某物时——或者对于我们来说，将选取器转动到某一个值——控件会告诉代理发生了什么事，而代理会判断如何做出反应。就像数据源一样，不同的控件也需要不同种类的代理。对于选取器，需要的是一个**UIPickerViewDelegate**。

有问必答

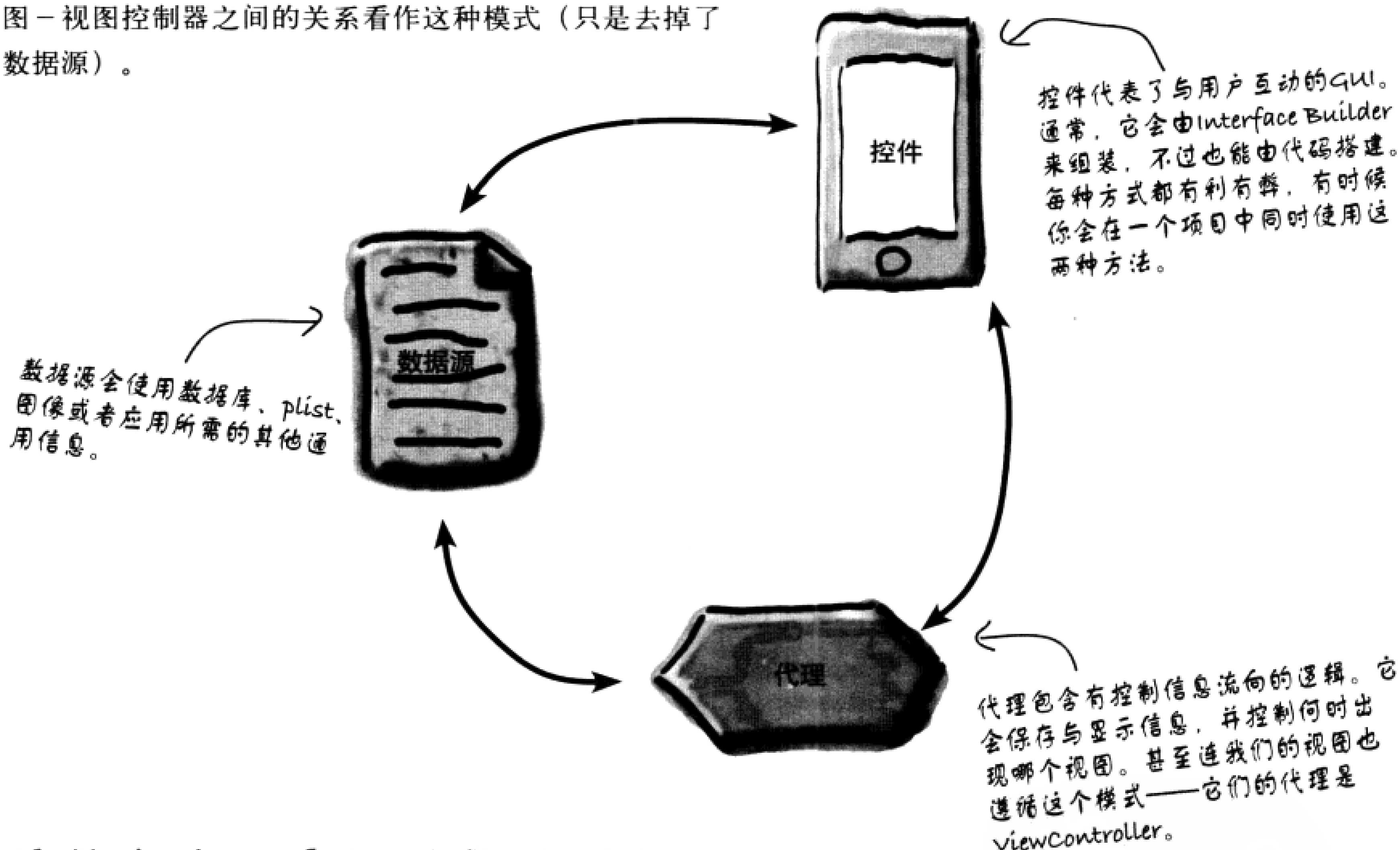
问： 为什么是代理提供内容呢？这看起来很像是数据啊。

答： 这是选取器的特殊情况，与选取器的代理可以改变数据的显示方式这一点有关。最简单的情况下，代理可以直接把字符串返回给选取器。如果它想更花哨一点，可以返回整个视图（对，就像你用Interface Builder搭建的视图一样，不过要小一点），那样就可以使用图像或者特殊字体之类的东西。

【拼吾爱-最新编程资源的分享下载站】

有一个模式是为此准备的

在这本书剩下的部分，你将会经常见到这个控件－数据源－代理模式。几乎所有的复杂控件都会使用它。 如果你不太较真的话，甚至可以把我们一直都在使用的视图－视图控制器之间的关系看作这种模式（只是去掉了数据源）。



控件有自己专门的数据源和代理

每种控件对于自己的数据源和代理都有特殊的要求，这里我们会简单讲一下，在Objective-C中是如何处理这种情况的。不过，重要的是要认识到，虽然在模式中，数据源和代理的职责是分离的，但并不一定要在不同的类中来实现它们。控件需要一个代理和一个数据源——它可不管它们是不是由同一个对象提供的：它只会要求数据源做数据源相关的工作，而代理做代理的活。

让我们仔细看看，UIPicker是怎么使用它的数据源和代理的，以便去理解这些东西是如何整合到一起的。

【拼吾爱-最新编程资源的分享下载站】



选取器揭密

本周访谈：
如何避免转得太快，超出控制……

Head First: 你好，选取器，谢谢你来到我们的节目现场。

选取器：这是我的荣幸。除了我的数据源和代理之外，一般没什么人和我说话，所以能到你们这儿来，我真是太高兴了。

Head First: 很高兴你能这么说。那么，我们曾经采访过其他一些控件，比如按钮和标签，不过他们都只拥有属性。这个代理和数据源是怎么一回事儿呢？

选取器：嗯，首先要澄清一点，我也拥有属性——只不过没什么出奇的地方罢了。令我与众不同的是处理大量数据的能力。我可以只有一行数据，也可以有上百行；这完全取决于应用程序。

Head First: 哦，是这么回事啊。一个标签里面只能有一个字符串，所以可以有一个属性，来保存字符串。我没说错吧。

选取器： 完全正确！所以，我并不是把所有数据都直接硬塞进肚里，而是在需要数据的时候再去请求数据，这样做更简单明了一些。

Head First: 但是你需要以一种特定的方式进行请求，是这样吗？

选取器：这正是我的设置中的魅力所在。我会以一种特殊的方式去请求我需要了解的内容——这就是为什么要有一个UIPickerDatasource——但是我不会去管数据源是从哪儿得到这些信息的。比如说，我需要知道要显示多少行，就去问我的数据源。数据源可能会使用数组、数据库或者plist，随便什么都行——我不管。我要知道的就是有多少行。

Head First: 这真不错——所以你可以显示来自任何来源的数据，只要你的数据源知道如何回答你的问题就行，而不必在乎它在内部是怎么存储这些数据的。

选取器：你说对了。现在看看代理，代理有点不太一样。转轮那些我都可以画出来，但我无法知道在有人选中一行时，每个应用程序都想要做什么，所以我直接把球踢给代理。

Head First: 因此不管是谁实现了代理，都得编一些代码，这样当你告诉他发生情况的时候，他可以执行正确的动作，比如保存一些值或者是设置时钟，诸如此类……

选取器：正是如此。在这里，我必须承认，在后面的步骤上我有一点小小的怪癖……

Head First: 哦？我洗耳恭听……这是说你需要知道某行要显示的值的时候，会向代理请求是吗？

选取器：是的——其他控件会向他们的数据源请求。我可以为此找出诸多借口，不过……算了，我们都会有点自己的癖好，对吧？

Head First: 我十分欣赏你的坦诚。不过，这样不完全是坏事；你的代理可以对每一行做一些处理，不是吗？

选取器： 哦，可不是！当我向代理要特定行时，他可以给回我一个完整的视图，而不仅仅是一个字符串。有的时候，他们可以在里面加上图表或者图像——真的，随便什么，只要能放进视图里，我就能显示。

Head First: 这挺不错。好，我们今天的时间就要到了，再次感谢你的光临。

选取器： 这是我的荣幸！现在我要去找我的新数据源，好好转转……

【拼吾爱-最新编程资源的分享下载站】

对号入座

将每个选取器特性与它所属的对象对应起来——数据源或是代理。你需要在API中深入挖掘，才能找出里面3个方法的来龙去脉。

选取器特性（或方法）	代理还是数据源？
指导如何为条目绘制视图	
部件的数目	代理
pickerView:numberOfRowsInComponent	
pickerView:titleForRow:forComponent	数据源
行中的值（字符串或视图）	
numberOfComponentsInPickerView	

通力合作之下，代理和数据源为选取器提供了显示自身所需的一切。

【拼吾爱-最新编程资源的分享下载站】

对号入座 答案

将每个选取器特性与它所属的对象对应起来——数据源或是代理。
你需要在API中深入挖掘，才能找出里面3个方法的来龙去脉。

选取器特性（或方法） **代理还是数据源?**

指导如何为项目绘制视图

代理

部件的数目

pickerView:numberOfRowsInComponent

UIPickerViewDataSource协议的必需部分，返回了行数。

pickerView:titleForRow:forComponent

数据源

UIPickerViewDelegate协议的一部分，会返回选取器中条目的标题。

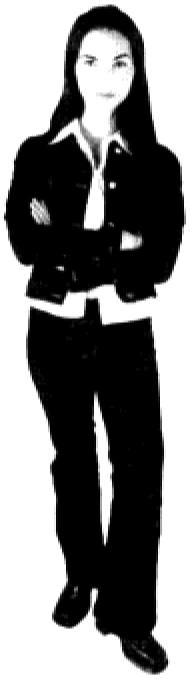
行中的值（字符串或视图）

numberOfComponentsInPickerView

UIPickerViewDataSource协议的必需部分，返回部件的数目。

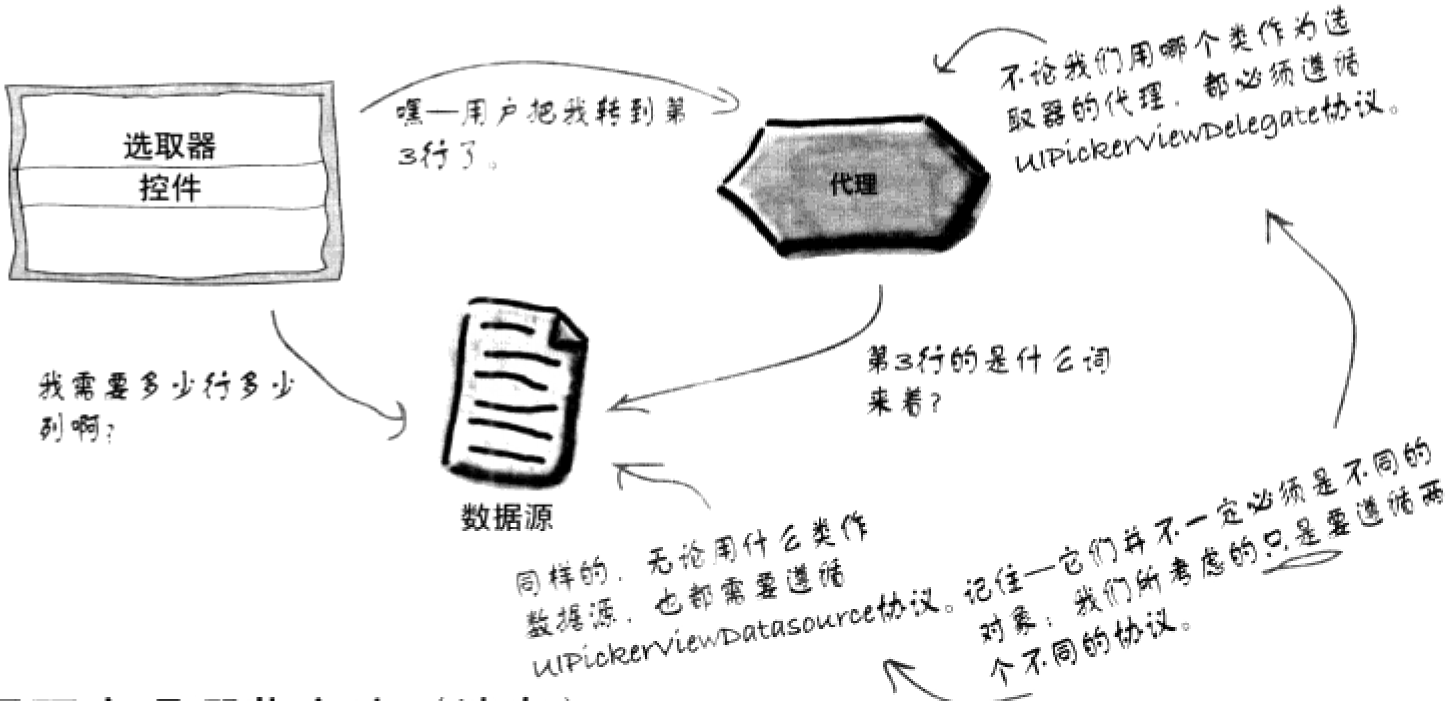
【拼吾爱-最新编程资源的分享下载站】





这些协议定义了数据源和代理需要对哪些消息做出响应。

选取器（以及其他使用代理和数据源的控件），指定了一些消息，要求他们的支持类对其进行响应。这些消息是以协议的形式来定义的。协议就是Objective-C中的纯虚接口。当你的类可以按照特定协议进行交流时，被称作遵循协议。



协议告诉你需要实现哪些方法（消息）

协议中通常会有一些方法必须要实现，而其他的方法则是可选的。例如，在UIPickerViewDatasource协议中，就有一个名为`pickerView:numberOfRowsInComponent`的必需方法；要想让选取器正常工作，数据源中就必须有这个方法。然而，UIPickerViewDelegate协议中就有一个可选方法，名为`pickerView:titleForRow:forComponent`，这个方法就不一定要出现在代理中，除非你要使用它。

那么如何得知需要考虑实现哪些协议呢？关于该元素的文档会告诉你它需要使用哪些协议。例如，我们的UIPickerView需要一个数据源，遵循的协议是UIPickerDataSource，还需要一个遵循UIPickerDelegate协议的代理。单击协议的名称，你就会看到关于协议的文档，说明哪些消息是可选的，哪些是必需的。在下一章中，会更详细地介绍如何实现这些协议；而目前，我们会直接把代码给你，好尽快开始项目。

【拼吾爱-最新编程资源的分享下载站】

首先，声明控制器遵循这两个协议

现在你已经了解到，要让选取器工作起来需要两样东西，也就是代理和数据源，让我们回到Xcode，将它们创建出来。你需要编辑Classes下面的两个文件：InstatwitViewController.h和InstatwitViewController.m。这两个文件都是在你项目开始的时候创建的。

这两个.h和.m文件是协同工作的，在头文件(.h)中声明了类的接口、变量原型、输出口以及动作方法等；实现文件(.m)中则是实际的实现代码。我们要更新头文件，声明我们的InstatwitViewController遵循 UIPickerViewDataSource和UIPickerViewDelegate这两个协议。

行动起来，把粗体部分添加进去。

```
#import <UIKit/UIKit.h>
@interface InstatwitViewController : UIViewController
<UIPickerViewDataSource, UIPickerViewDelegate> {
        NSArray* activities;
        NSArray* feelings;
    }
@end
```

在此声明我们的类将遵循UIPickerViewDataSource和UIPickerViewDelegate协议。

我们会为迈克设立两个数组：一个用于活动，一个用于感觉。

InstatwitViewController.h

接下来，在实现文件里加入迈克的活动和感觉

现在我们要开始修改InstatwitViewController.m文件了，这是真正实现功能的部分。需要加入一些方法，来实现协议的要求，但是稍后我们再进行此事。首先让我们来得自迈克的列表添加进去。我们会利用头文件里声明的两个数组，来存储迈克给我们的那些词组。

所有实现代码都放在@implementation后面。这里我们表明要实现的是在头文件里定义的InstatwitViewController接口。

```
#import "InstatwitViewController.h"
@implementation InstatwitViewController
```

这里打断了，跳过了部分注释掉的代码，我们不会用到这些默认生成的代码。

InstatwitViewController.m

【拼吾爱-最新编程资源的分享下载站】

把此处原有的/*记号去掉，然后加入下面的代码。这个方法属于视图控制器，会在视图从.xib文件中载入后被调用。在这里你可以为视图做一些初始化与设置工作。

```
// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
 - (void)viewDidLoad {
    [super viewDidLoad];
        activities = [[NSArray alloc] initWithObjects:@"睡觉", @"吃饭", @"干活", @"思考", @"哭泣", @"乞求", @"离去", @"购物", @"问候世界", nil];
        feelings = [[NSArray alloc] initWithObjects:@"心情不错", @"悲哀", @"快乐", @"心情矛盾", @"恶心", @"神经兮兮", @"困惑", @"满怀希望", @"焦虑", nil];
}
```

在此我们用还充的列表建立了两个数组，稍后我们会将其填入选取器中。

字符串前面的"@"，是用来告诉编译器将它们作为NSString，而不是char*来对待。NSString是真正的Objective-C类，而后者只是一个简单的C风格的字符指针。大多数Objective-C类都会使用NSString，而不用char*。



InstatwitViewController.m

```
- (void)dealloc {
    [activities release];
    [feelings release];
    [super dealloc];
}
@end
```

你得释放所有对象，来清理内存，因为iPhone是小型设备（所以内存不是很多）。在第3章中我们会好好说一说内存问题。

现在万事俱备，只欠协议了……

【拼吾爱-最新编程资源的分享下载站】

数据源协议有两个必需的方法

首先，让我们关注数据源协议的方法。在头文件里，我们声明了InstatwitViewController遵循UIPickerViewDatasource协议。这个协议中有两个方法是必需的，分别是：numberOfComponentsInPickerView:pickerView和pickerView:numberOfRowsInComponent。我们知道在视图里需要两个转轮（部件），因此可以首先把这个方法放到我们的实现文件中：

这里是选取器的两个必需方法。

```
 - (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)
pickerView {
        return 2;
 }
 - (NSInteger)pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent :(NSInteger)component {
        if (component == 0) {
                return [activities count];
        }
        else {
                return [feelings count];
        }
}
```

有多少部件？

每个部件里有多少行？这个数字来自不同的数组，所以我们需要分别对待。



InstatwitViewController.m

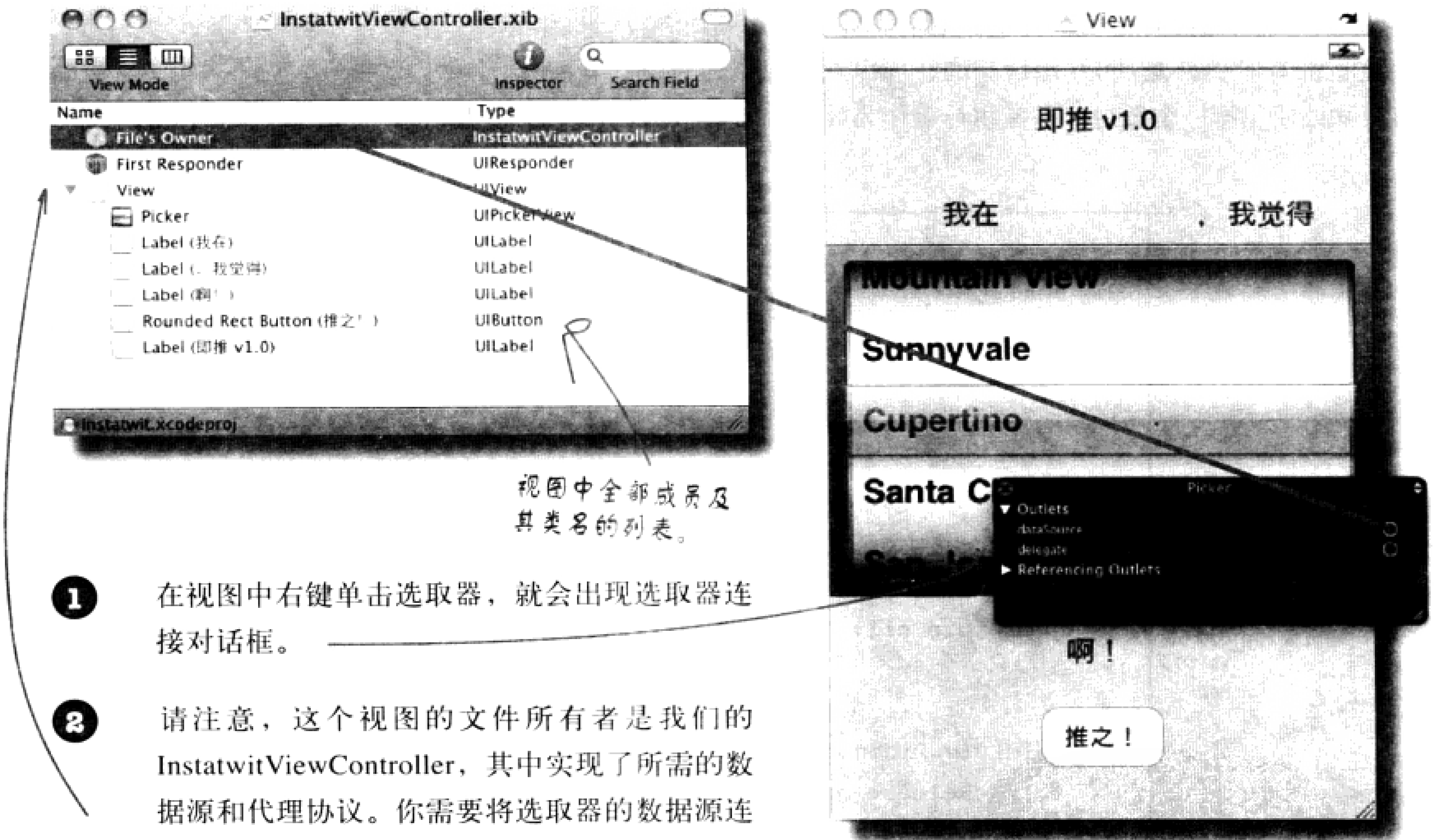
第二个方法需要返回每个部件中的行的数目。参数component会告诉我们选取器正在询问的是哪一个部件，第一个部件（活动）是部件0。每个部件中的行数就是相应数组中项目的个数。

现在已经实现了方法，让我们将其与选取器连接到一起。

【拼吾爱-最新编程资源的分享下载站】

连接数据源与连接动作方法和输出口类似

既然数据源协议已经实现完毕，数据业已就位，剩下的事就是把它和选取器联系起来了。跳回到Interface Builder中，建立起联系：



❶ 在视图中右键单击选取器，就会出现选取器连接对话框。

❷ 请注意，这个视图的文件所有者是我们的InstatwitViewController，其中实现了所需的数据源和代理协议。你需要将选取器的数据源连接到我们的控制器，也就是文件所有者。为此，要在数据源旁边的圆圈内部单击，并拖曳到文件所有者。



如果在Interface Builder中没有保存，将不会有任何效果！

Xcode只会使用最后保存下来的版本，其他任何东西都不起作用。

接着是代理……

【拼吾爱-最新编程资源的分享下载站】

代理协议中只有一个必需方法

UIPickerViewDelegate协议仅仅有一个必需方法（嗯，从技术角度来说，其实还有两个可选方法，而且你必须实现其中的一个）。我们打算使用pickerView:titleForRow:forComponent。这个方法必须返回一个NSString，其中包含有给定部件的标题。和前面的情况一样，这些值也都有索引，是从0开始计算的，所以我们可以通过部件值得知该使用哪一个数组，并使用行数作为数组中的索引。

这些消息的原型来自UIPickerViewDelegate和UIPickerViewDataSource的文档。你愿意的话，可以直接复制粘贴过来。

```
- (NSString *)pickerView:(UIPickerView *)pickerView
titleForRow:(NSInteger)row forComponent:(NSInteger)component {
    switch (component) {
        case 0:
            return [activities objectAtIndex:row];
        case 1:
            return [feelings objectAtIndex:row];
    }
    return nil;
}

- (void)viewDidUnload {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}
```

我们选择了实现两个方法，这是其中一个。

返回数组中相应位置的字符串——第0行对应第一个字符串，第1行是第二个，以此类推。

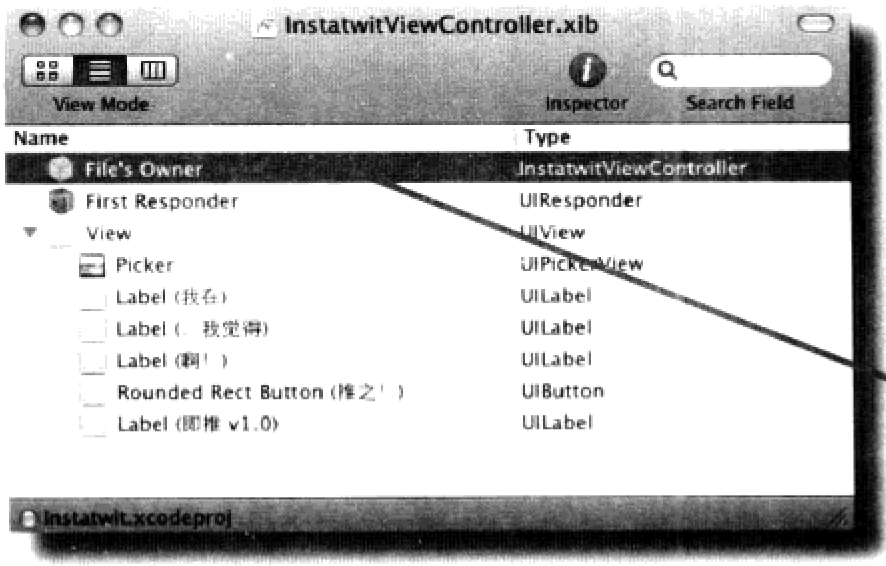
这会在你的应用关闭、视图被卸载时被调用。我们目前还不需要用它，所以就让它按照模板原样不动。

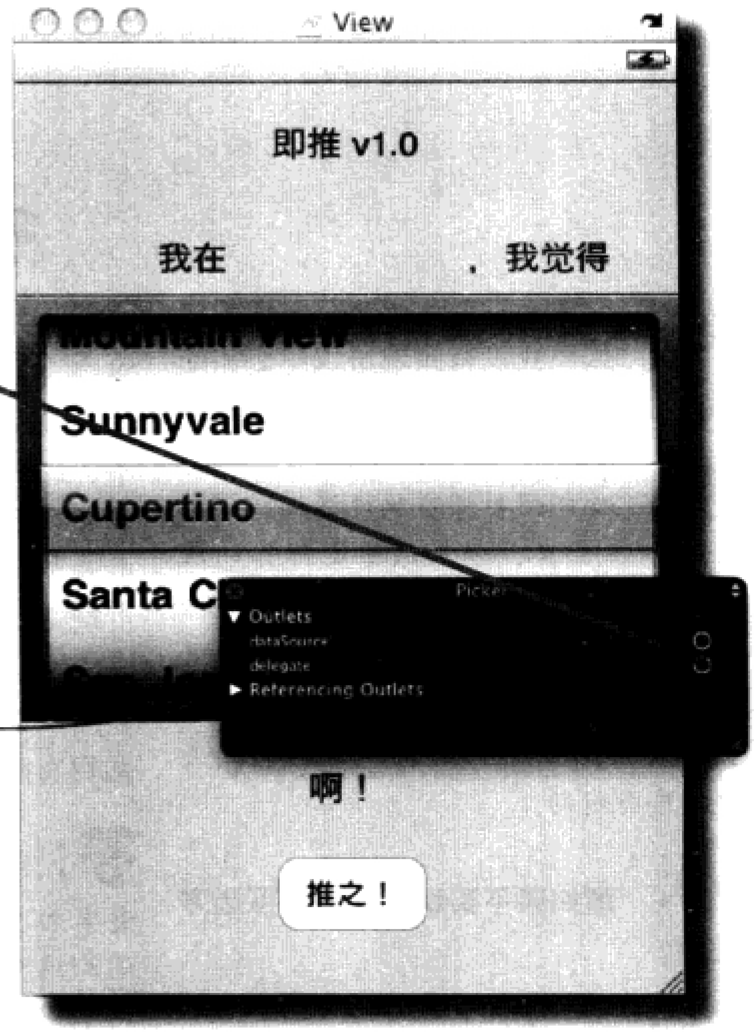


InstatwitViewController.m

现在回到Interface Builder去连接代理……

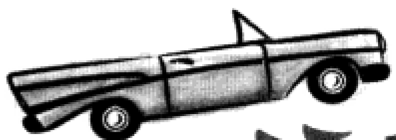
【拼吾爱-最新编程资源的分享下载站】





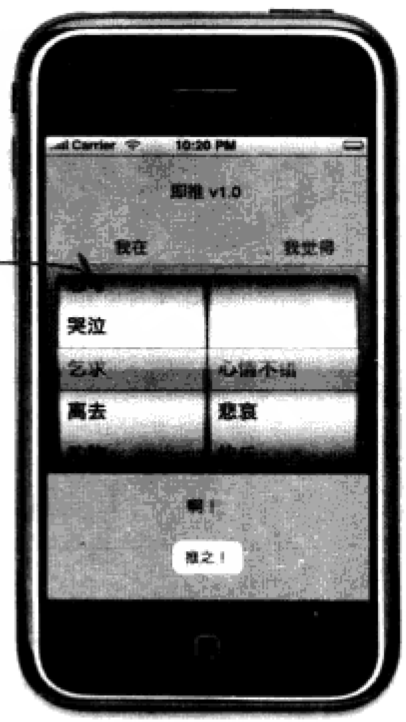
❶ 再次右键单击选取器，让连接窗口出现。

❷ 文件所有者同样也实现了代理协议。单击代理旁边的圆圈，拖曳到文件所有者。



运行测试

在Interface Builder中保存你的工作成果，回到Xcode中，同样保存，然后点击Build and Run（或者按⌘+回车）。模拟器弹出之后，你就会看到全部成果了！



拨动这些转轮——迈克的列表里面的内容全都在这儿了，而且工作得很好！

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：如果我没有实现协议中必需的方法会怎么样？

答： 你的项目可以编译通过，不过会有一些警告。如果试图运行你的应用程序，它几乎百分之百地会出现一个“unrecognized selector”异常，并引起程序崩溃，这是由于部件试图向你的类发送必需的消息，而消息在类中缺失引起的。

问：那如果不实现协议中的可选方法呢？

答： 这没关系。不过这些方法所提供的任何功能都无法使用。对此你必须小心斟酌，因为有时候苹果公司会把几个方法标记为可选，但是你必须至少实现其中的一个。UIPickerViewDelegate就遇到了这种情况。如果文档中指出的方法你一个都不实现，应用在运行的时候就会出现一个错误，并随之崩溃。

问：对于一个类能够实现的协议的数目有限制吗？

答： 没有。不过，你实现的越多，在类中加入的代码也就越多，所以总会遇到这种情况，你必须得把功能分拆到不同的类当中去，才能正常地管理代码。不过从技术角度来说，你想实现多少协议都可以。

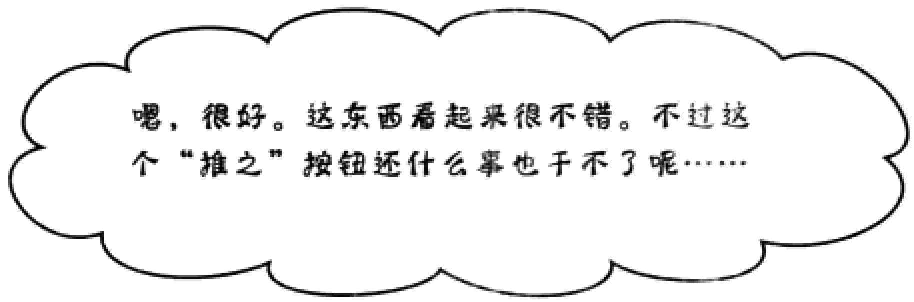
问：我有点晕了，我们放进头文件里的接口和协议中的有什么分别啊？

答： 头文件中的接口是Objective-C用来声明一个类会对哪些属性、字段和消息做出响应的。它就像C++中的头文件或者Java文件的方法声明一样。但是，你必须为类接口中的一切提供实现代码。另一方面，协议则只是消息的列表——根本没有实现。要靠完成了协议的类为之提供实现。这些协议相当于Java中的接口和C++中的纯虚方法。

关键要点

- 选取器需要一个代理和一个数据源才能正常工作。
- 在选取器中，每个转轮都是一个部件。
- 在选取器中，每个条目是一行。
- 协议定义了你的类必须实现的消息——其中一些可能是可选的。

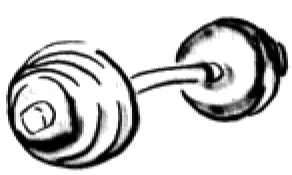
【拼吾爱-最新编程资源的分享下载站】





现在我们开始让这个按钮与Twitter对话……

我们已经让选取器工作起来了，不过如果你试一下“推之”按钮，就会发现虽然选中了内容，但什么事也没发生。这个按钮仍然有待于我们的进一步改进，令其对迈克作出反应，然后把所有内容发送给Twitter。



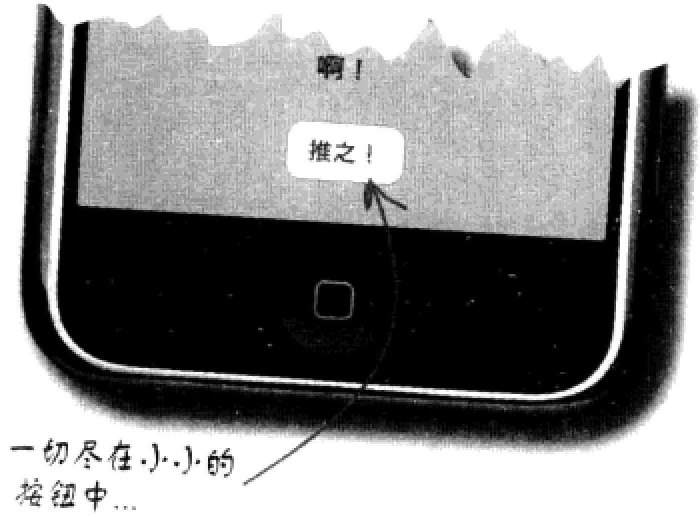
脑力举重

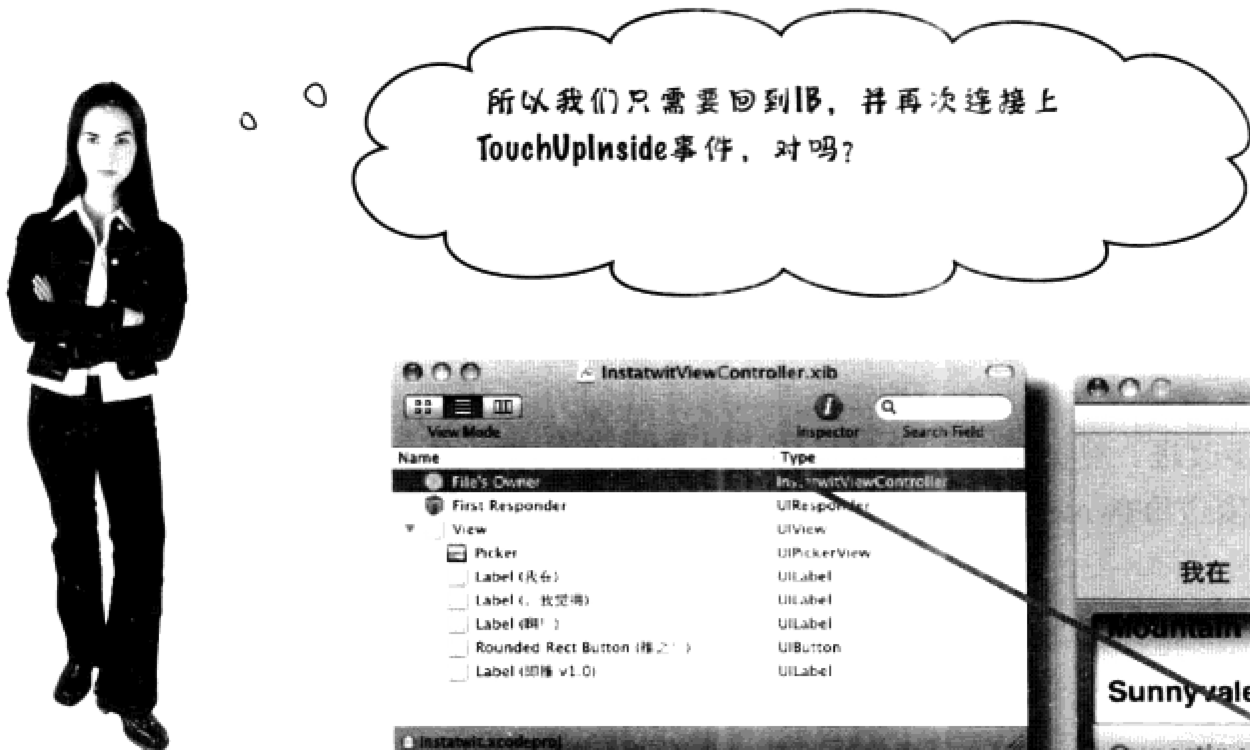
想一想要让按钮正常工作，我们需要做些什么？将要使用哪些文件？按钮实际上要做哪些事？

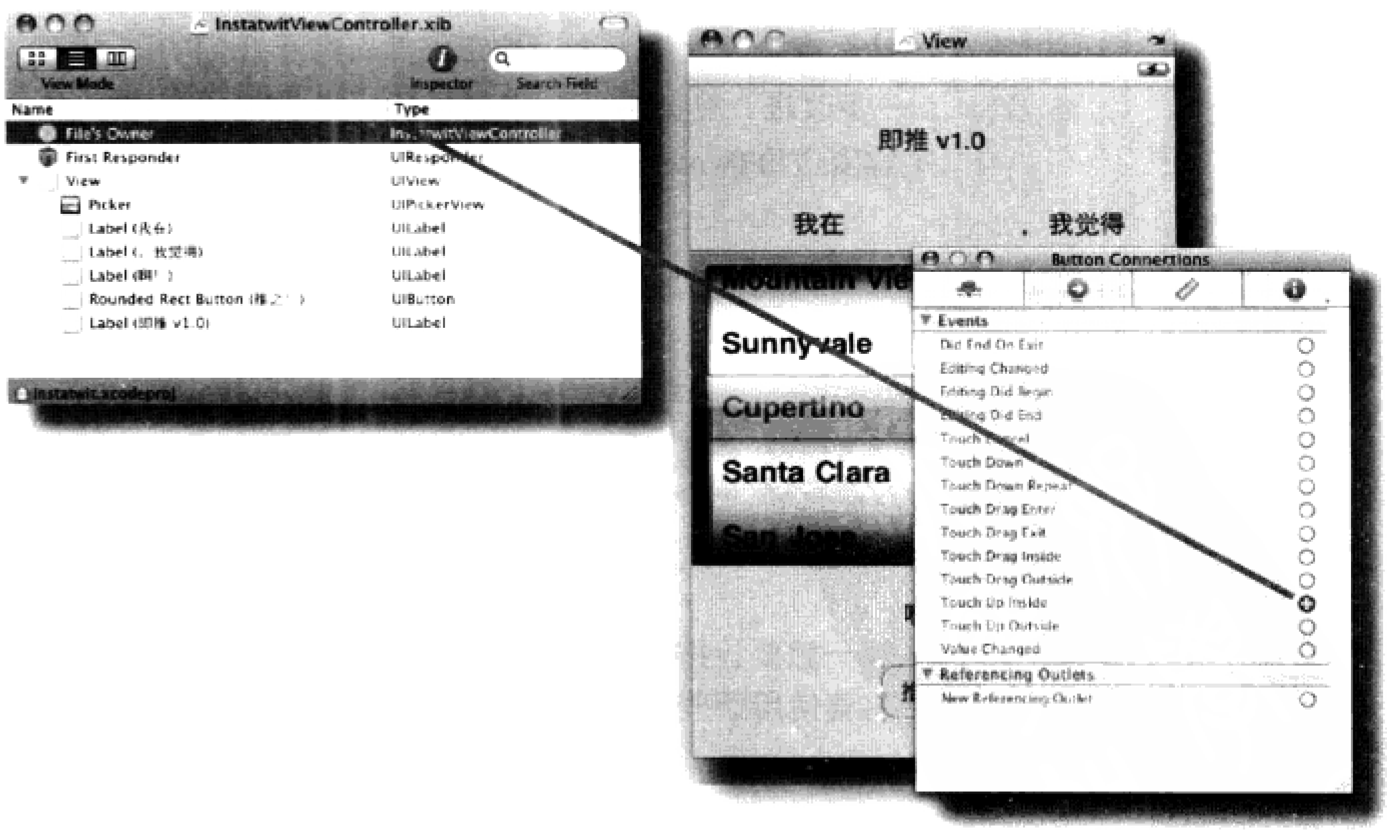
【拼吾爱-最新编程资源的分享下载站】

按钮需要连接到事件

我们需要把按钮组装起来，就像在第1章中那样。在迈克选好他在做什么和对此的感觉之后，就会点击“推之！”然后我们需要从选取器获得他的选择，并发送到Twitter。







是的，不过我们把这个事件连接到何处呢？

【拼吾爱-最新编程资源的分享下载站】

没有动作方法，你的按钮将无法工作！

我们在第1章中已经学习过动作方法，没有它的话，在Interface Builder的连接窗口中将无法连接到任何东西。

这是我们在第1章中为按钮创建的按下动作方法：

```
- (IBAction) buttonPressed:(id)sender;
```

所有IBAction消息都接收一个参数：消息的发送者。也就是触发动作方法的界面元素。

IB = Interface Builder

这是将要被调用的方法的名字。这个名字可以随便取，但是方法必须有一个类型为id的参数。



练习

我们需要同时修改InstatwitViewController的头文件和实现文件。

1. 先从头文件开始，在里面加入一个IBAction，命名为sendButtonTapped。

2. 然后在.m文件中为这个方法提供一个实现，并在里面向日志中写入一条信息，这样可以在实际向Twitter发送之前，就得知方法已正常工作。

【拼吾爱-最新编程资源的分享下载站】



练习答案

在头文件中声明你的IBAction，并在.m文件中提供实现。

1

```
#import <UIKit/UIKit.h>
@interface InstatwitViewController : UIViewController
<UIPickerViewDataSource, UIPickerViewDelegate> {
        NSArray* activities;
        NSArray* feelings;
}
- (IBAction) sendButtonTapped: (id) sender;
```

IBAction让代码对用户事件做出响应，记得吗……

在这儿声明你的IBAction，我们就可以在.m文件中使用它，Interface Builder也可以得知我们提供了一个动作方法。



InstatwitViewController.h

2

```
- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
  [super didReceiveMemoryWarning];
    // Release any cached data, images, etc that aren't in use.
}

- (IBAction) sendButtonTapped: (id) sender {
    NSLog(@"Tweet button tapped!");
}
```

与.h中声明的方法相同

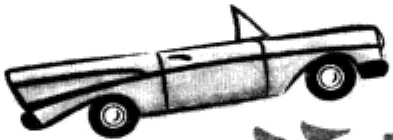
这可以让你在控制台上看到输出…



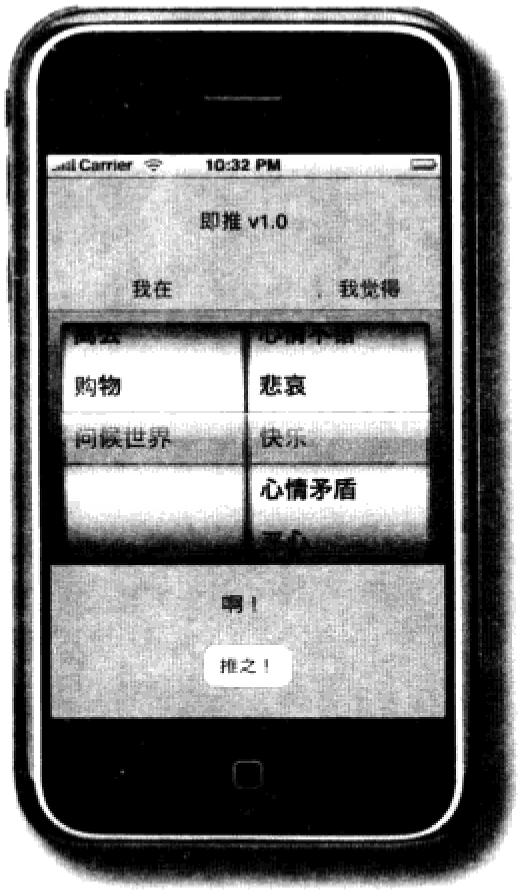
InstatwitViewController.m

现在回过头去用Interface Builder把它搭起来……

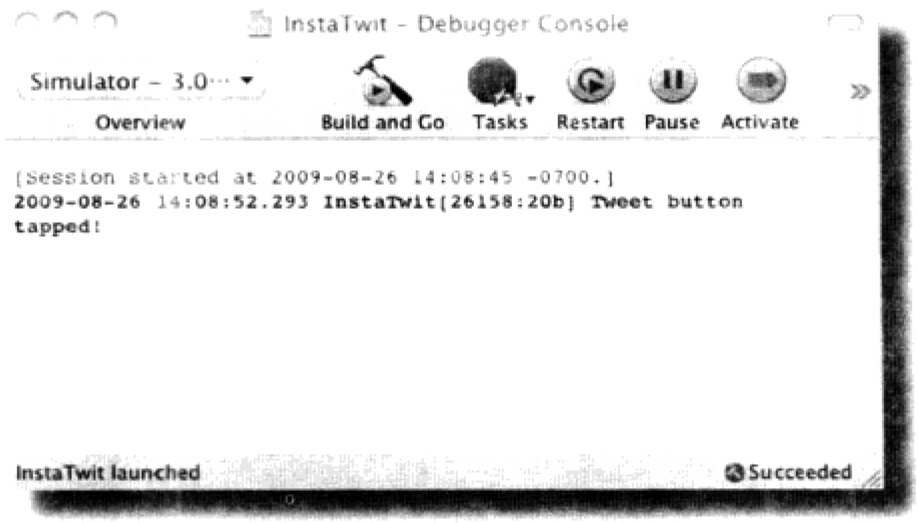
【拼吾爱-最新编程资源的分享下载站】

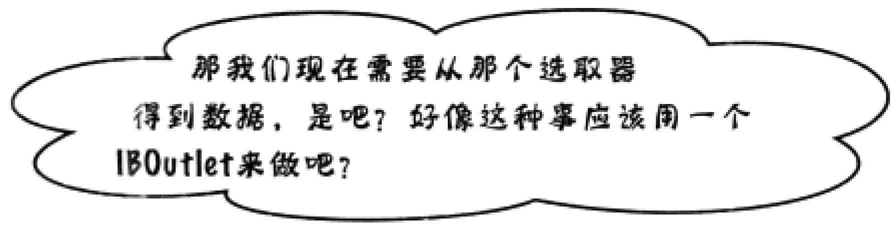


运行测试



保存，然后单击Build and Run。你会在控制台中看到一条信息，“Tweet button tapped!”







是的！IBOutlet提供了对选取器的引用。

在第1章中，我们使用了一个输出口来访问并修改按钮上的文本字段的值。现在，为了收集要实际发送到Twitter的消息，我们需要从选取器中抽取出被选中的值，然后按照选中的标签文字创建一个同样内容的字符串。

到目前为止，都是选取器在需要信息的时候呼叫我们；这次，当迈克点击了“推之”按钮后，轮到我们向选取器要数据了。我们会用一个IBOutlet来完成此事。

【拼吾爱-最新编程资源的分享下载站】

在我们的视图控制器中添加IBOutlet和属性

除了要声明IBOutlet之外，我们还得声明一个同名属性。在下一章里我们会仔细说说属性，不过简单来说，它可以帮助我们正确地管理内存，并在nib加载时，让Cocoa Touch框架为我们设置tweetPicker字段的值。

先从头文件开始……

```
#import <UIKit/UIKit.h>
@interface InstatwitViewController : UIViewController
<UIPickerViewDataSource, UIPickerViewDelegate> {
    IBOutlet UIPickerView *tweetPicker;
    NSArray* activities;
    NSArray* feelings;
}
@property (nonatomic, retain) UIPickerView* tweetPicker;
- (IBAction) sendButtonTapped: (id) sender;

@end
```

这儿我们在类中声明了一个字段，名为tweetPicker。其类型是一个指向UIPickerView的指针。



InstatwitViewController.h

这儿是我们的输出口声明。这使得Interface Builder可以得知有一些对象要进行连接。IBOutlet是用#defined定义的，实际上没有任何内容，只是给Interface Builder用的。

对应tweetPicker的属性有一些内存管理的选项，我们会在第3章中对此做出详细解释。

【拼吾爱-最新编程资源的分享下载站】

……然后加入实现。

```
#import "InstatwitViewController.h"
@implementation InstatwitViewController
@synthesize tweetPicker;
```



InstatwitViewController.m

@synthesize总是与.h文件中的@property相伴随。更多信息见第3章…

```
- (void)dealloc {
      [tweetPicker release];
      [activities release];
      [feelings release];
    [super dealloc];
}

@end
```

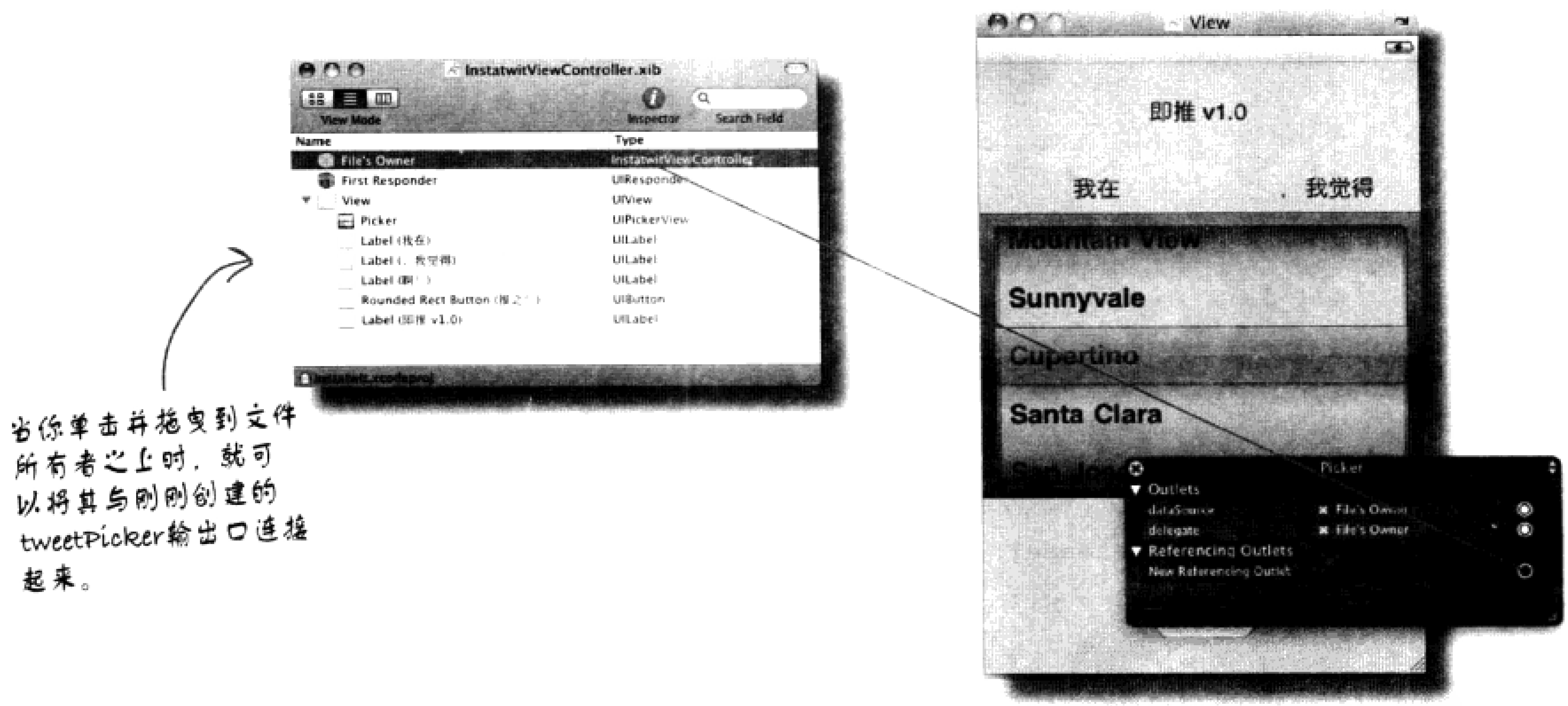
关于tweetPicker的最后一项工作，是要将对它的引用释放掉——又是一桩内存事宜。我们会在第3章补上关于内存管理的这一课，我们向毛主席保证。

接下来做什么？

【拼吾爱-最新编程资源的分享下载站】

连接选取器和输出口

你可能对此期待已久了！回到Interface Builder，建立UIPickerView与我们的视图控制器中的IBOutlet的连接。右键单击UIPickerView，拖曳“New Referencing Outlet”字样旁边的圆圈，放在文件所有者上——InstatwitViewController会显示出它新建的tweetPicker输出口。



脑力练习

现在你需要做什么，才能获得选取器的数据并将其放入你的Twitter消息中？考虑一下“推之！”按钮的按下动作方法，想想该怎么改……

【拼吾爱-最新编程资源的分享下载站】

通过选取器的引用获取选中值

现在剩下的唯一工作，就是通过我们持有的对选取器的引用，获取迈克实际选择的内容。我们需要重新实现sendButtonTapped方法，从选取器中将值抽取出来。查看UIPickerView的文档，会发现我们需要使用的方法是selectedRowInComponent:。这个方法会返回选中行的数值表示，同前面一样，这个数值可以在我们的数组里用作索引。

这就是我们实现的回调函数。我们需要创建一个字符串，并在其中填上从选取器中得到的值。格式字符串中的"%@"会被我们传入的值代替。

要得知迈克在选取器中选中的内容，需要询问选取器哪一行被选中，并从数组中取得相应的字符串。

```
- (IBAction) sendButtonTapped: (id) sender {
 NSString* themessage = [NSString stringWithFormat:@"我在%@，我觉得%@啊！",
          [activities objectAtIndex:[tweetPicker selectedRowInComponent:0]],
          [feelings objectAtIndex:[tweetPicker selectedRowInComponent:1]]];
    NSLog(themessage);
    NSLog(@"Tweet button tapped!");
}
```

将这条日志信息去掉，换成最终的Twitter消息。

我们想打造一个新的字符串，里面有完整的推文内容，所以使用了NSString的stringWithFormat方法，来通过模板创建一个字符串。在格式字符串中可以使用多种占位符，比如字符、整数等，但是目前我们仅仅需要插入两个选中的字符串，因此使用了%@。



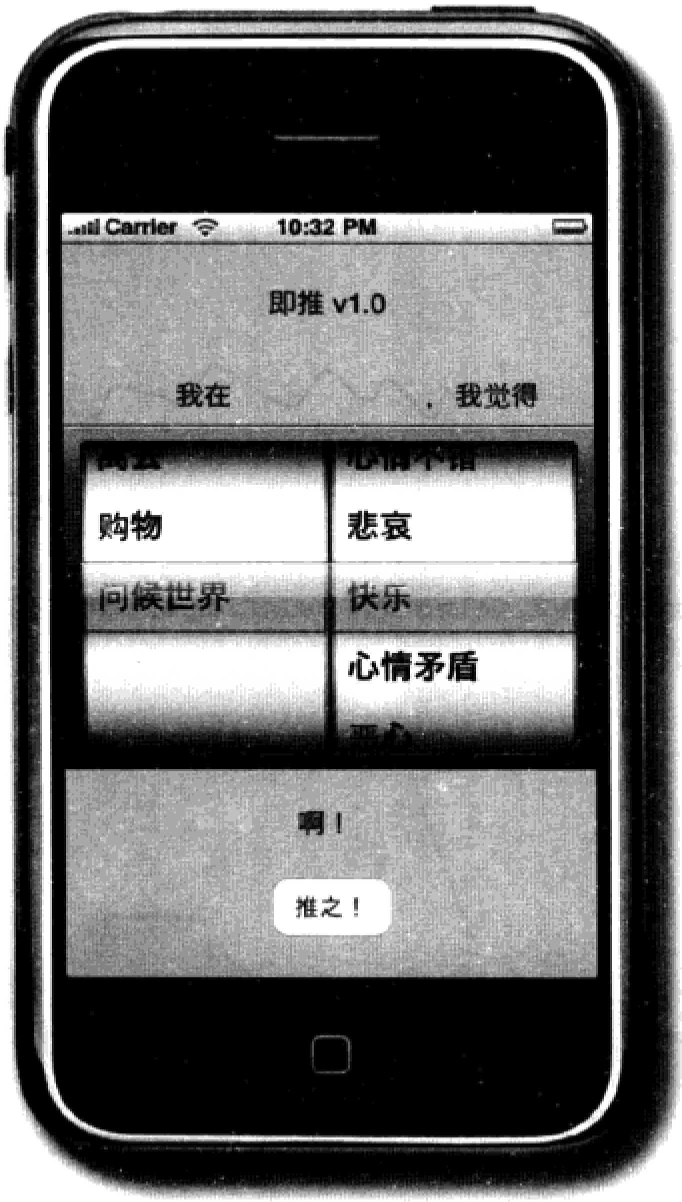
InstatwitViewController.m

我们打算把这条消息作为日志显示在控制台上，这样就可以看到构建的字符串，稍后我们就会将其发送到Twitter。但首先我们还是先确认这个方法已经正确实现，然后再向全世界发推……

【拼吾爱-最新编程资源的分享下载站】

运行测试

好了，现在我们试一下。你应该在控制台中见到一条十分正宗的推文：





把Twitter信息加上以后，这就是将会实际显示在你的馈源（feed）中的推文。

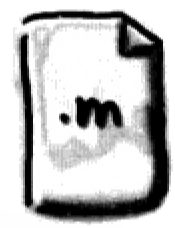
现在只剩下与Twitter通讯了——我们会帮你完成它。



要在Twitter上发帖，就得使用他们的API。我们并不想把本书变成Twitter API教程，所以会直接给出发推所需的代码。请把下面的代码输入到InstatwitViewController.m中，放在sendButtonTapped方法里，紧挨着输出Twitter消息的NSLog语句的下面。

需要将你的用户名和密码写在这里。

```
//TWITTER BLACK MAGIC
        NSMutableURLRequest *theRequest=[NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@”http://YOUR_TWITTER_USERNAME:YOUR_TWITTER_PASSWORD@twitter.com/
statuses/update.xml”]
                        cachePolicy:NSURLRequestUseProtocolCachePolicy
                        timeoutInterval:60.0];
        [theRequest setHTTPMethod:@”POST”];
        [theRequest setHTTPBody:[[NSString stringWithFormat:@”status=%@”,
themessage] dataUsingEncoding:NSASCIIStringEncoding]];
        NSURLResponse* response;
        NSError* error;
        NSData* result = [NSURLConnection sendSynchronousRequest:theRequest
returningResponse:&response error:&error];
        NSLog(@”%@”, [[[NSString alloc] initWithData:result
encoding:NSASCIIStringEncoding] autorelease]);
//      END TWITTER BLACK MAGIC
```


InstatwitViewController.m



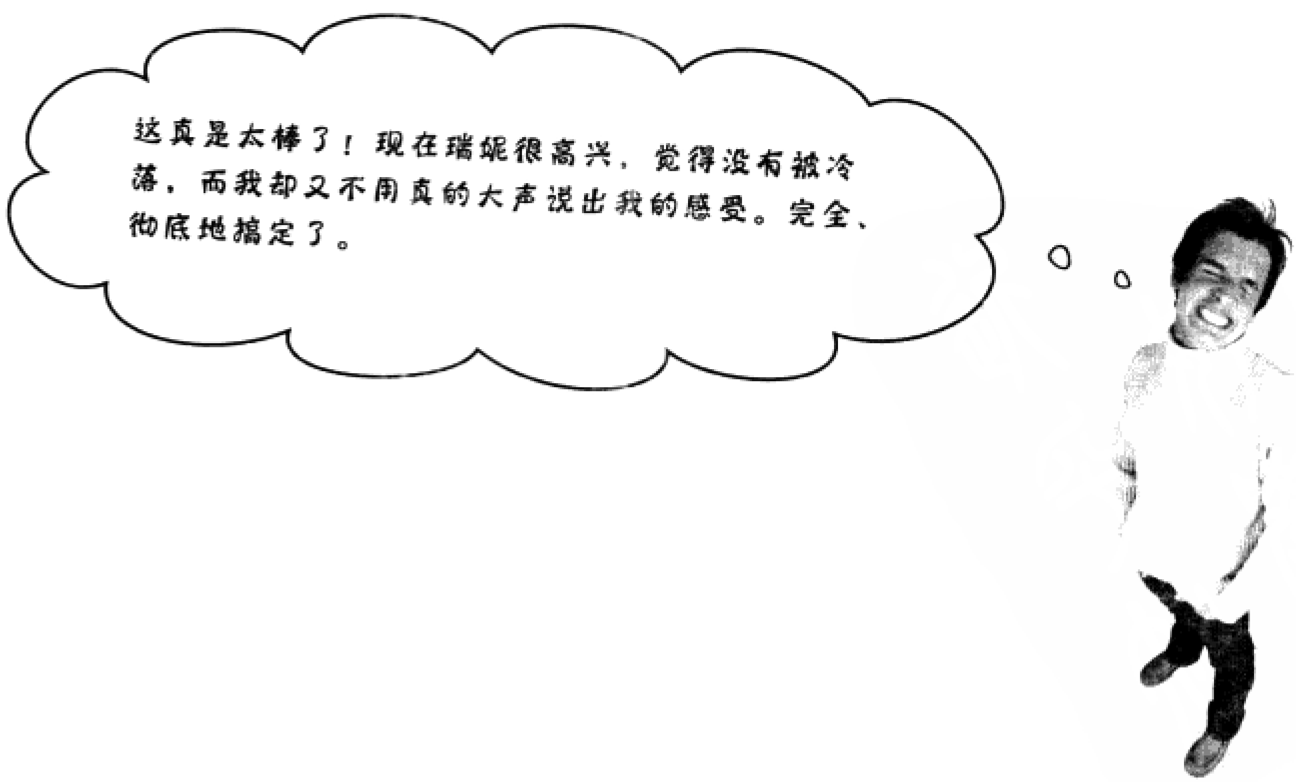
如果你还没有Twitter账号，赶快去注册一个！

只要来到twitter.com并注册即可。完成之后，你就可以输入用户名和密码，这个应用就会运行无碍了（译注：得翻墙）。

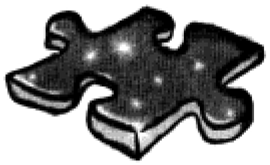
添加这些代码之后，你可以保存，然后编译并执行。信息就会出现在Twitter馈源中。行动起来，尝试一下。

【拼吾爱-最新编程资源的分享下载站】



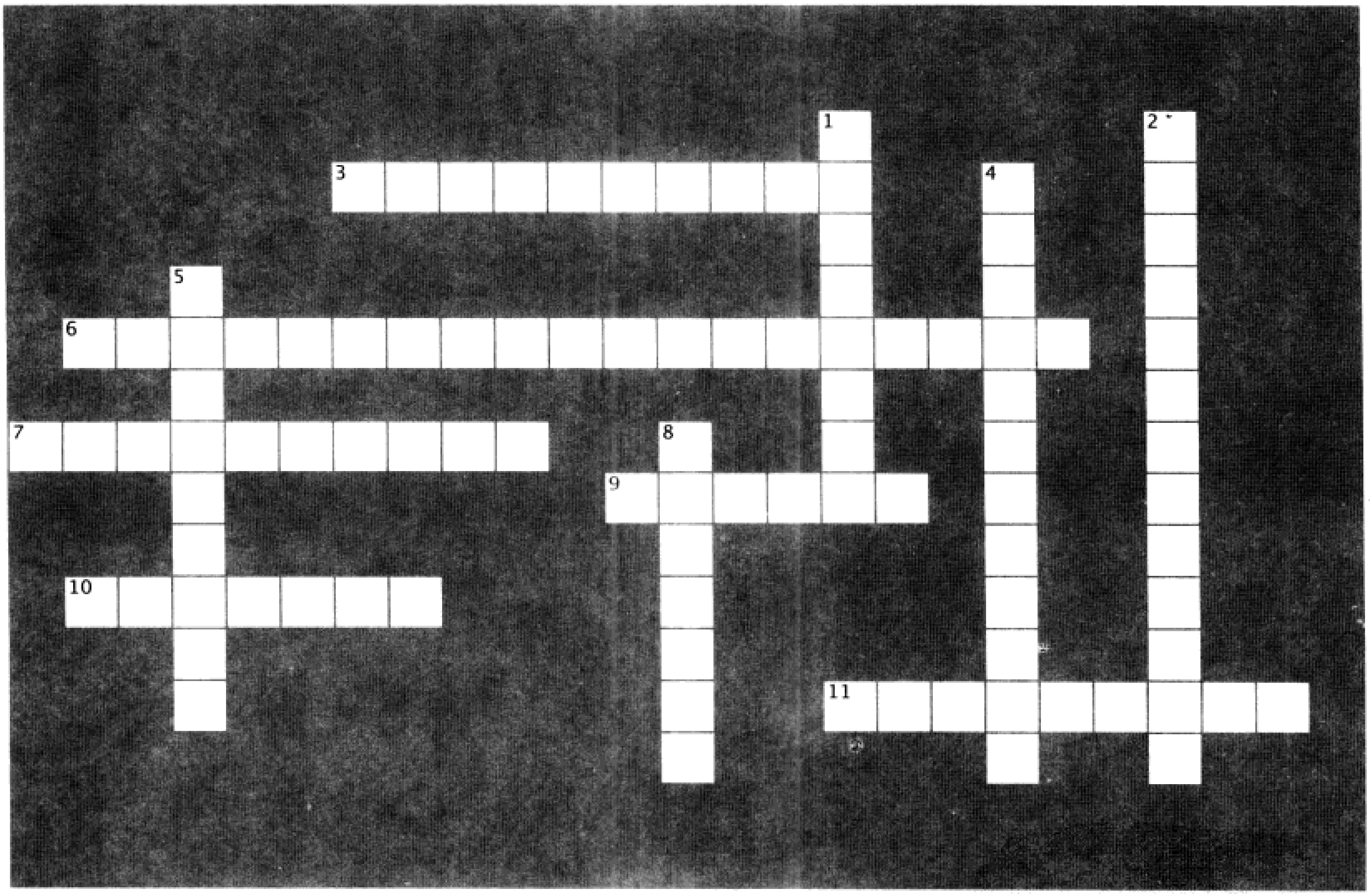


【拼吾爱-最新编程资源的分享下载站】



iPhone纵横字谜

用字谜来提高你运用词汇的技巧。



横向提示

3. 此物通常用来处理应用中自身的信息。
6. 苹果公司用来衡量App Store中应用的文档。
7. 你会在视图中见到此物被列出，它能控制视图。
9. 这种组件可以控制只从一些选定的值中获得需要的输入。
10. 这种类型的应用通常只有一幅画面，只提供一些最基本的交互功能。
11. 定义了数据源和代理要对哪些消息进行响应。

纵向提示

1. 此物通常包含控制应用内信息流向的逻辑。
2. 要找出需要实现哪些协议，最好的办法就是查看______。
4. 这种类型的应用通常会涉及具有结构层次的数据。
5. 这种类型的应用大多数使用自定义的控制器和图形。
8. *.xib文件的另一个名字。

【拼吾爱-最新编程资源的分享下载站】



我们列出了对几个不同应用的描述。请根据这些描述，勾勒出一个大致的视图，并回答关于每个应用的问题。

1 **通用巨型按钮应用**

现在已经有几个这种应用在App Store中发售了。这个应用的使用方式是按下一个巨大的按钮，并从你的iPhone中发出一些噪音。

这个应用是什么类型的？

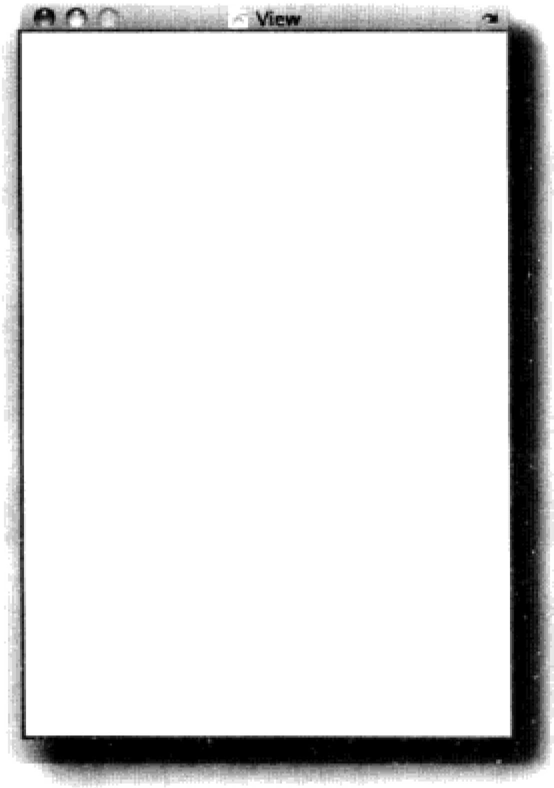
....

对于这种应用类型，HIG中最关注的方面是什么？

....

....

....



2 **书籍清单应用**

这个应用的目的是帮你管理图书馆中的藏书列表，并随之提供简要的说明，介绍书籍的题材与作者。

这个应用是什么类型的？

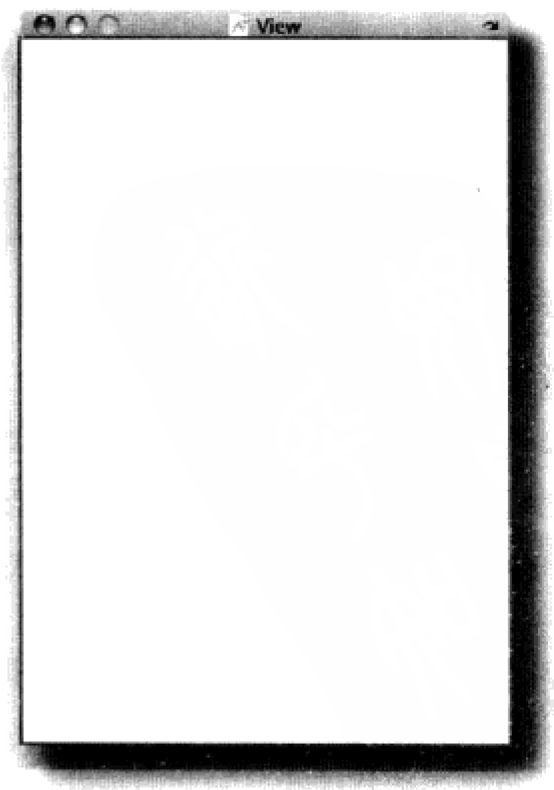
....

对于这种应用类型，HIG中最关注的方面是什么？

....

....

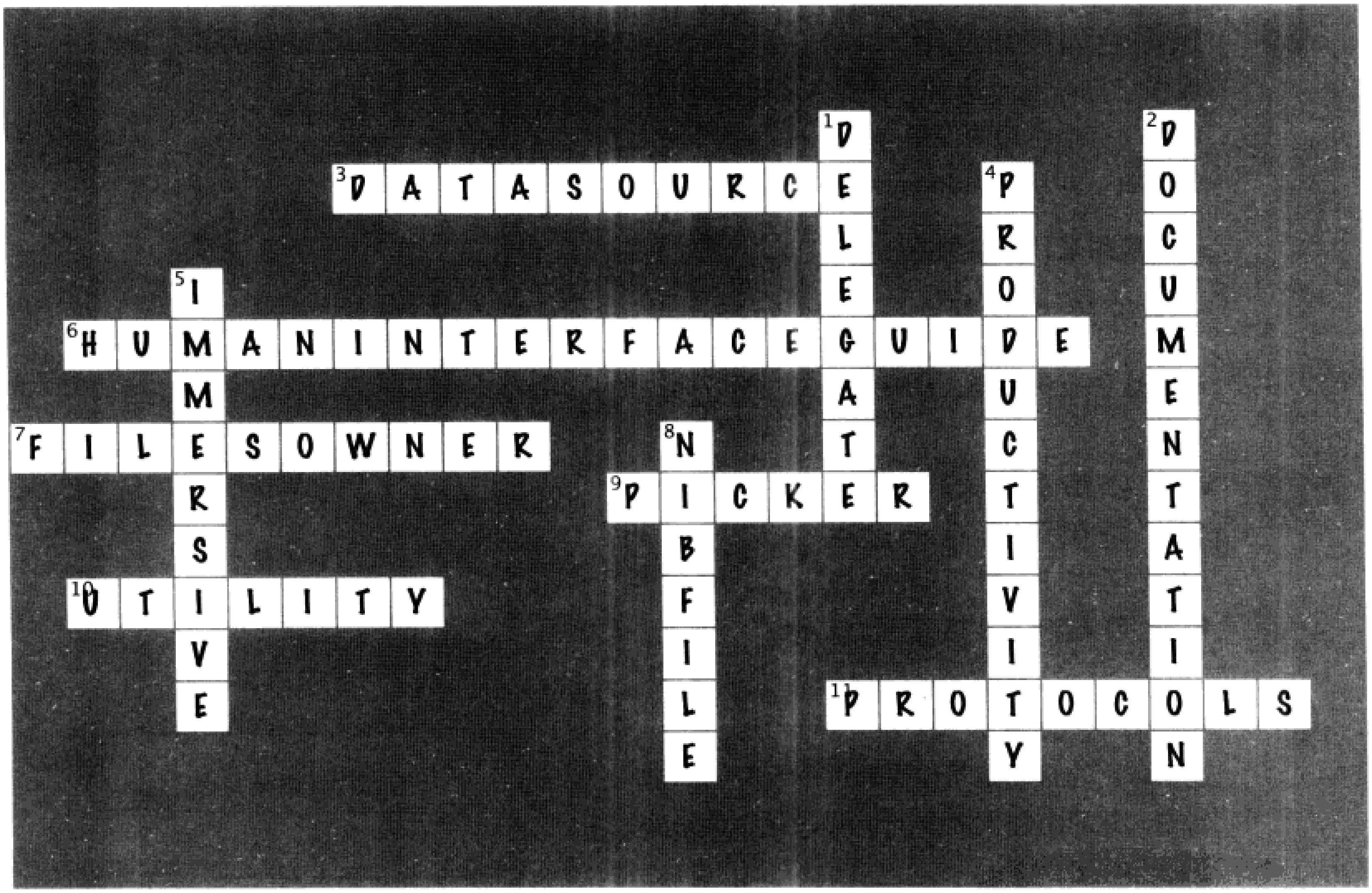
....



【拼吾爱-最新编程资源的分享下载站】

iPhone纵横字谜答案

用字谜来提高你运用词汇的技巧。



横向提示

3. 此物通常用来处理应用中自身的信息。[DATASOURCE]
6. 苹果公司用来衡量App Store中应用的文档。[HUMANINTERFACEGUIDE]
7. 你会在视图中见到此物被列出，它能控制视图。[FILESOWNER]
9. 这种组件可以控制只从一些选定的值中获得需要的输入。[PICKER]
10. 这种类型的应用通常只有一幅画面，只提供一些最基本的交互功能。[UTILITY]
11. 定义了数据源和代理要对哪些消息进行响应。[PROTOCOLS]

纵向提示

1. 物通常包含控制应用内信息流向的逻辑。[DELEGATE]
2. 要找出需要实现哪些协议，最好的办法就是查看__________。[DOCUMENTATION]
4. 这种类型的应用通常会涉及具有结构层次的数据。[PRODUCTIVITY]
5. 这种类型的应用大多数使用自定义的控制器和图形。[IMMERSIVE]
8. *.xib文件的另一个名字。[NIBFILE]

【拼吾爱-最新编程资源的分享下载站】

练习答案

我们列出了对几个不同应用的描述。请根据这些描述，勾勒出一个大致的视图，并回答关于每个应用的问题。

❶ 通用巨型按钮应用

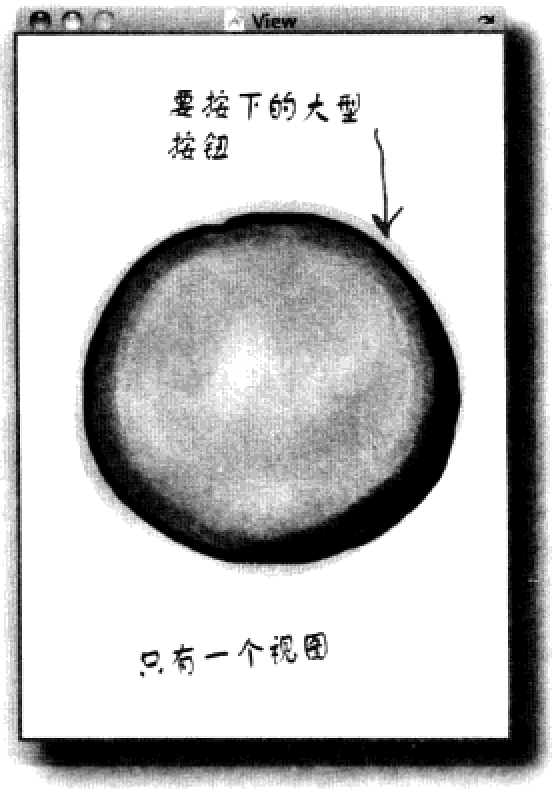
现在已经有几个这种应用在App Store中发售了。这个应用的使用方式是按下一个巨大的按钮，并从你的iPhone中发出一些噪音。

这个应用是什么类型的？

一个沉浸式应用

对于这种应用类型，HIG中最关注的方面是什么？

苹果公司最关心的问题就是控件都“提供了内在一致的体验”。所以任何东西都可以定制，这种应用需要目标专注并组织良好。



❷ 书籍清单应用

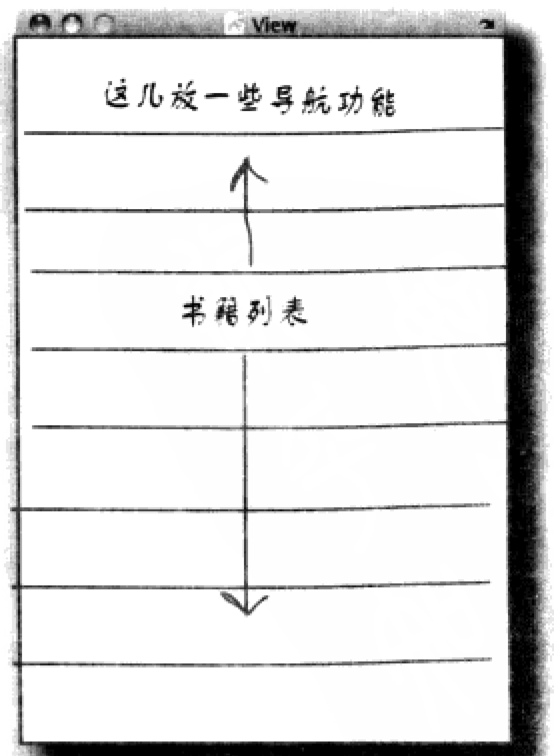
这个应用的目的是帮你管理图书馆中的藏书列表，并随之提供简要的说明，介绍书籍的题材与作者。

这个应用是什么类型的？

一个生产力应用

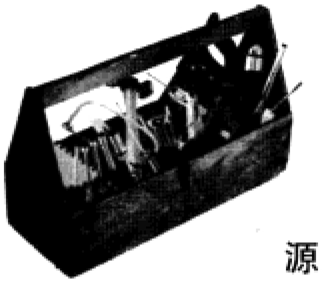
对于这种应用类型，HIG中最关注的方面是什么？

对于这种应用类型，因为你要使用标准控件，HIG有许多更详尽的规则。每个控件都需要被检查，以确保使用正确。



另一个视图用于显示细节，得想办法才能用到它...

【拼吾爱-最新编程资源的分享下载站】



你的iPhone工具箱

你已经消化吸收了第2章，现在你的工具箱中又增加了协议、代理和数据源。如果你想得到一份完整的书中功能提示的列表，请访问http://www.headfirstlabs.com/iphonedev。

代理

负责UI元素的行为举止…

包含有控制信息流向的逻辑，比如保存或者显示数据，以及何时显示哪个视图，等等。

可以与数据源位于同一对象内，但自身有专门的协议。

协议

定义了你的数据源和代理必须响应的消息。

在头文件（.h）中声明。

协议中某些内容可能会是可选项。

数据源

为控件与控件要显示的数据之间提供一座沟通的桥梁。

可以使用数据库、plist、图像以及其他一些应用需要显示的通用信息。

可以与代理位于同一个对象中，但自身有专门的协议。

关键要点

- 选取器需要一个代理和一个数据源才能正常工作。
- 选取器中每个转轮都是一个部件。
- 选取器中每个选项都是一行。
- 协议定义了你的类必须实现的消息——其中一些可能是可选的。

【拼吾爱-最新编程资源的分享下载站】



看来迈克将会需要对InstaTwit做一些修改，才能让他的恋爱关系继续稳固发展……

【拼吾爱-最新编程资源的分享下载站】

3 面向iPhone的Objective-C教程

Twitter需要多姿多彩



在第2章中我们颇写了不少东西，但是用的那是什么语言啊？写的这些代码中，有些部分你可能有点眼熟，不过也是时候揭开谜底，让你明白这到底是怎么回事了。iPhone SDK自带了一些非常棒的工具，让你不必事事都要写代码来实现，但是如果你不学习一下底层使用的语言，不了解属性、消息传递以及内存管理，还是无法编写一个完整的应用的。除非你能做到全部应用都只使用默认的组件！而你是希望能做到更多的事情的，对吧？

【拼吾爱-最新编程资源的分享下载站】

瑞妮有点明白过来了……

迈克小心翼翼地用着InstaTwit来交流感情，但是他的女友开始觉得有点不对劲儿了。就算迈克这家伙总是喜欢按部就班，他的推文看起来也有点让人起疑了。

InstaTwit工作非常良好，而且也很易于使用！不过我觉得瑞妮盯上我了。她说我好像总是说同样的话。我得能往我的推文里加点东西，不然这样下去就要露馅儿了。



我们需要对InstaTwit的设计做出一些调整。

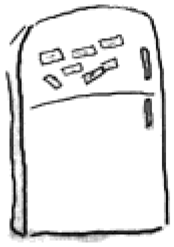
看一看Interface Builder里各种各样的UI控件吧，再想一想有没有什么好办法，能让迈克快捷又方便地在他的推文里加上一些内容。

【拼吾爱-最新编程资源的分享下载站】

给自定义输入腾地方

不必施展什么奇招妙计，只要用一个文本字段，迈克就可以在他的推文里加一点个人特色。这意味着他得打几个字，不过结果是他的推文将会变得独特一些。





设计魔力贴

使用你在为程序加入选取器和按钮时学到的知识，将下面的魔力贴与添加文本字段时要修改的方法或者文件对应起来。

1. 在InstatwitViewController.h中 .... 。

2. 在InstatwitViewController.m的顶部 .... 。

3. 在InstatwitViewController.m的`dealloc`方法中， .... 。

4. 使用Interface Builder， .... 。

5. 对于第1步中创建的属性，使用 Interface Builder， .... 。

为notesField创建代理和数据源

在视图中加入一个UITextField

为UITextField添加一个IBOutlet和@property声明

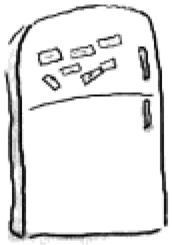
为UITextField添加一个IBAction

将notesField加入@synthesize的行列

添加[notesfield release]

将UITextField与IBOutlet连接起来

【拼吾爱-最新编程资源的分享下载站】



设计魔力贴答案

使用你在为程序加入选取器和按钮时学到的知识，将下面的魔力贴与添加文本字段时要修改的方法或者文件对应起来。

❶ 在InstatwitViewController.h中 ……为UITextField添加一个IBOutlet和@property声明……。

```
IBOutlet UIPickerView *tweetPicker;
    IBOutlet UITextField *notesField;
    NSArray* activities;
    NSArray* feelings;
}

@property (nonatomic, retain) UIPickerView* tweetPicker;
@property (nonatomic, retain) UITextField* notesField;
```

我们需要的就是一个UITextField。要实现这个新的字段元素，我们需要声明一个类成员（加上IBOutlet，这样Interface Builder可以识别它），并加入一个属性，命名为notesField。



InstatwitViewController.h



等一下。我们一直在往这个.h文件中添加代码，可是我还没弄明白.h文件到底是干什么用的呢！到底怎么回事啊？

.h文件就是头文件。

这就是你为一个类声明接口和方法的地方。到目前为止，我们用到的所有的类，比如UITextField、NSString和NSArray等，都有自己的头文件。你可以花一点时间，好好研究几个头文件，也开始思考一下这些文件中的内容。



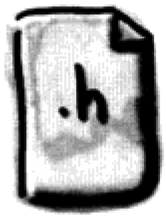
留神私有框架的头文件

有时你会不经意地发现一个非常诱人的方法，却没有在苹果公司的文档中定义。如果使用了未公开的API，会使得你的应用被iTunes store拒绝。

【拼吾爱-最新编程资源的分享下载站】

头文件为你的类描述接口

在Objective-C中，类是在头文件中通过接口来定义的。在这里可以声明你的类是否继承自某个基类，也可以声明类的成员变量、属性和方法。



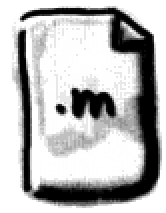
InstatwitViewController.h

接口、类变量、方法和属性

```
@interface InstatwitViewController :
UIViewController
```

```
IBOutlet UIPickerView *tweetPicker;
```

```
- (IBAction) sendButtonTapped: (id) sender;
```



InstatwitViewController.m

拿起笔来

下面是我们当前的InstatwitViewController.h文件。请在空白处填写内容，解释每一行代码的功能。

```
#import <UIKit/UIKit.h>

@interface InstatwitViewController : UIViewController
<UIPickerViewDataSource, UIPickerViewDelegate> {
        IBOutlet UIPickerView *tweetPicker;
        IBOutlet UITextField *notesField;
        NSArray* activities;
        NSArray* feelings;
}

@property (nonatomic, retain) UIPickerView* tweetPicker;
@property (nonatomic, retain) UITextField* notesField;

- (IBAction) sendButtonTapped: (id) sender;
- (IBAction) textFieldDoneEditing:(id) sender;

@end
```



InstatwitViewController.h

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

下面是我们当前的InstatwitViewController.h文件。请在空白处填写内容，解释每一行代码的功能。

```
#import <UIKit/UIKit.h>
```

#import将另一个文件（绝大多数情况是一个头文件），在编译时集成到当前文件中。这种方法被用来从其他文件中引入类、常量等。

除了会自动地防止将同一头文件包含多次之外，这与C的#include几乎完全一样（因此不再需要使用#ifndef MY_HEADER）。

@interface表明打算声明一个类。

紧接着就是类的名称，如果继承自其他类，再加上一个冒号和父类的名字。

Objective-C不支持多继承。

这是我们的继承关系和接口。

```
@interface InstatwitViewController :
UIViewController <UIPickerViewDataSource,
UIPickerViewDelegate> {
```

你打算实现的协议，都用尖括号括起来，用逗号分隔。协议就像是Java的接口或者C++中的纯虚类，一个类可以实现任意多个协议。

IBOutlet让Interface Builder可以识别你要与控件联系在一起的成员变量（比如我们的InstaTwit里的notesField）。

这是我们声明类成员变量的地方。

```
    IBOutlet UIPickerView *tweetPicker;
    IBOutlet UITextField *notesField;
    NSArray* activities;
    NSArray* feelings;
}
```

声明成员变量的语法与C++类似：int和float这样的基本类型可以直接使用；指针类型通过星号来表示。默认情况下，所有成员变量的访问权限都是protected，但与C++类似，可以指定@private或@public段落对此进行修改。



InstatwitViewController.h

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

搞定了接口中成员变量的部分以后，你就可以声明属性了。@property标记会让Objective-C自动为你生成读写方法。

这些是属性的特性：很快我们就会对此进一步展开说明…

```
@property (nonatomic, retain) UIPickerView* tweetPicker;
```

关键字@property告诉编译器这是一个属性，需要有对其进行读写（也许只读）的方法。

这是类型和属性名，与类中的成员变量相同。

```
@property (nonatomic, retain) UITextField* notesField;
```

减号代表这是一个实例方法（加号表示静态方法）。Objective-C中的所有方法都是公有的。

这些是方法声明。

```
- (IBAction) sendButtonTapped: (id) sender;
- (IBAction) textFieldDoneEditing:(id) sender;
```

IBAction令Interface Builder能够识别哪些方法可以与事件关联起来。

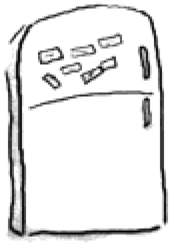
IBAction方法必须接受一个类型为id的参数，这个类型就类似于C++中的void *，或者是Java中的Object。

```
@end
```

@end：表示你的类接口声明到此结束。


InstatwitViewController.h

【拼吾爱-最新编程资源的分享下载站】



设计魔力贴答案（续）

回到我们正在进行的设计工作...

使用你在为程序加入选取器和按钮时学到的知识，将下面的魔力贴与添加文本字段时要修改的方法或者文件对应起来。

❶ 在InstatwitViewController.h中 **为UITextField添加一个IBOutlet和@property声明**。

```
        IBOutlet        UIPickerView *tweetPicker;
    IBOutlet UITextField *notesField;
    NSArray* activities;
    NSArray* feelings;
}
@property (nonatomic, retain) UIPickerView* tweetPicker;
@property (nonatomic, retain) UITextField* notesField;
```



InstatwitViewController.h

❷ 在InstatwitViewController.m的顶部 **将notesField加入@synthesize的行列**。

```
@synthesize tweetPicker, notesField;
```



InstatwitViewController.m

在此你为@property合成访问方法。你可以重新添加一行@synthesize代码，或者直接把属性加在现有的行中，用逗号分隔。

OK，也就是说我们在.h文件里声明一个属性，然后在.m文件里加入@synthesize，就一定会自动生成一些代码，对吗？



正是如此！这样会生成出读写方法。

编译器从`@property`可以知道有一个属性，但这还不够。在实现文件中用`@synthesize`关键字对其进行合成，就会让编译器自动生成我们之前谈到过的读写方法。编译器一定会生成读方法，如果属性是`readwrite`的，也会生成写方法，并且会根据.h文件中声明的@property中的特性将这些方法实现。那么，不同的@property特性都有什么作用呢……？

【拼吾爱-最新编程资源的分享下载站】

对号入座

下面列出的是属性最常用的特性及其定义。请将特性与定义对应起来。

readonly	让属性可以被别人修改。编译器会为之分别生成读写方法各一。这是默认值。
retain	用于基本类型的属性，比如int、float等。编译器会为其创建一个简单的写入方法，仅使用如myField = value这样的语句来赋值。这也是默认值，不过常常并非你所愿。
readwrite	用于值类型为对象的属性。编译器会持有（retain）你传入的值（我们马上会进一步解释“持有”），并在有新值传入时将原有值释放。
copy	用于不想让其他人修改属性的情况。你仍然可以修改属性背后的成员变量的值，但编译器不会生成写入方法。
assign	用于希望保存一份传入值的拷贝而不是值自身的情况；例如，你要保存一个数组，而在将这个数组设置为属性值后，又不想让其他人再修改其内容。这会给传入的值发送一个copy消息，然后持有复制的结果。

【拼吾爱-最新编程资源的分享下载站】

对号入座答案

下面列出的是属性最常用的特性及其定义。请将特性与定义对应起来。

readonly

retain

readwrite

copy

assign



让属性可以被别人修改。编译器会为之分别生成读写方法各一。这是默认值。

用于基本类型的属性，比如int、float等。编译器会为其创建一个简单的写入方法，仅使用如myField = value这样的语句来赋值。这也是默认值，不过常常并非你所愿。

用于值类型为对象的属性。编译器会持有（retain）你传入的值（我们马上会进一步解释“持有”），并在有新值传入时将原有值释放。

用于不想让其他人修改属性的情况。你仍然可以修改属性背后的成员变量的值，但编译器不会生成写入方法。

用于希望保存一份传入值的拷贝而不是值自身的情况；例如，你要保存一个数组，而在将这个数组设置为属性值后，又不想让其他人再修改其内容。这会给传入的值发送一个copy消息，然后持有复制的结果。

有问必答

问： 编译器怎么能知道要用哪个成员变量来保存属性的值呢？

答： 默认情况下，编译器会假定属性的名字与成员变量的名字相同。在实际使用中，这也不是必需的。你可以在用@synthesize合成属性时，指定支持它的成员变量，比如：@synthesize secretString=_superSecretField;。

问： 那nonatomic这个关键字是什么意思呢？

答： 默认情况下，生成的访问方法都是线程安全的，会在修改属性值的时候使用mutex加锁。这种操作方式被称为原子性（atomic）的。然而如果你的类并不会被多个线程使用，这么做就是一种浪费。你可以将属性声明为非原子的（nonatomic），从而告诉编译器忽略mutex加锁机制。

【拼吾爱-最新编程资源的分享下载站】

自动生成的访问方法也要进行内存管理

iPhone上的Objective-C没有垃圾收集器（gc），所以你必须使用引用计数来管理内存。引用计数会记录对一个对象有多少引用，只有在计数降为0（就是不再被使用）时，才会将空间释放。在使用属性时，编译器会替我们处理这个问题。目前为止我们声明的属性都使用了retain特性。当编译器为这样的属性生成写入方法时，就会准确地进行内存管理，如下所示：

Nonatomic意味着不上锁…

Retain说明我们使用的是对象类型，而且我们要保有传给写入方法的对象。

```
@property (nonatomic, retain) NSString* secretString;
@synthesize secretString
```

这应该出现在@implementation部分。

这里编译器仅仅返回成员变量的值，没什么新鲜的。

```
- (NSString*) secretString {
        return secretString;
 }
```

因为我们没说属性是只读的，所以编译器就会为其生成一个写入方法。

```
- (void) setSecretString: (NSString*) newValue {
        if (newValue != secretString) {
                [secretString release];
                secretString = [newValue retain];
        }
 }
```

由于使用了retain关键字，生成的写入方法会检查新值，确保与老值不是同一个对象，然后释放老值，持有新值。

拿起笔来

写出Objective-C会为下列属性声明生成的代码。

```
1. @property (nonatomic, readonly) NSString* myField

2. @property (nonatomic, retain) NSString* myField

3. @property (nonatomic, assign) NSString* myField
```

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

写出Objective-C会为下列属性声明生成的代码。

1. `@property (nonatomic, readonly) NSString* myField`

```
- (NSString*) myField {
 return myField;
}
```

2. `@property (nonatomic, retain) NSString* myField`

```
- (NSString*) myField {
  return myField;
}
- (void) setMyField: (NSString*)
newValue {
  if (newValue !=myField) {
    [myField release];
      myField = [newValue retain];
  }
}
```

3. `@property (nonatomic, assign) NSString* myField`

```
- (NSString*) myField {
  return myField;
}
 - (void) setMyField: (NSString*) newValue
  {
  myField = newValue;
}
```

小心这个东西… NSStrings是使用引用计数的对象，所以虽然这样写从技术上是可行的，但对一个NSString属性使用赋值（assign），则很可能会有问题。

不过对于基本类型，比如boolean和float，是不能使用引用计数的。绝大多数情况下，赋值都是合适的用法。

【拼吾爱-最新编程资源的分享下载站】



我猜，这个release调用就释放了你的属性所占用的内存空间，对吗？

Objective-C也可以自动释放引用。

除了持有与释放之外，Objective-C还有一个内存管理概念，就是自动回收池。基本上，这就是一个装满对象的数组，运行库会在处理完当前事件时，调用这些对象的release方法。想把对象放到自动回收池中很简单，只要向它发送autorelease消息即可：

```
[aString autorelease];
```

该对象被持有的引用计数会保持不变，但是在当前消息循环结束后，它会收到一个release消息。这么做有一定的性能开销，没有直接释放来得高效，所以可能你不会在所有场合下使用这种方法。使用自动回收机制也不错，不过最好还是尽可能地去显式持有和释放对象。

要让内存保持一致，你只需要记住两件事

明白吗？内存，还记得吗？

在较大型的应用中，内存管理可能会变得非常繁琐，所以苹果公司为此设立了几条规则，来明确谁来负责持有和释放对象以及何时进行。

1 用alloc、new、copy或者mutableCopy创建的对象，必须用release释放。

如果用alloc、new、copy或者mutableCopy创建了一个对象，它的引用计数就会是1，而你不再使用该对象时，得负责向其发送一个release消息。如果你希望稍后由系统来发送释放消息，也可以将对象放进自动回收池当中。

2 对于其他任何对象，都可以认为引用计数为1，同时处于自动回收池中。

如果你通过任何其他方式（字符串格式化生成器、数组初始化等等）得到了一个对象，都可以将它看作是一个引用计数为1且已经位于自动回收池中的对象。也就是说，如果你想要在获得这个对象的方法之外仍持有它，就得向它发送一个retain消息（相应地，以后还要发送一个release）。

【拼吾爱-最新编程资源的分享下载站】

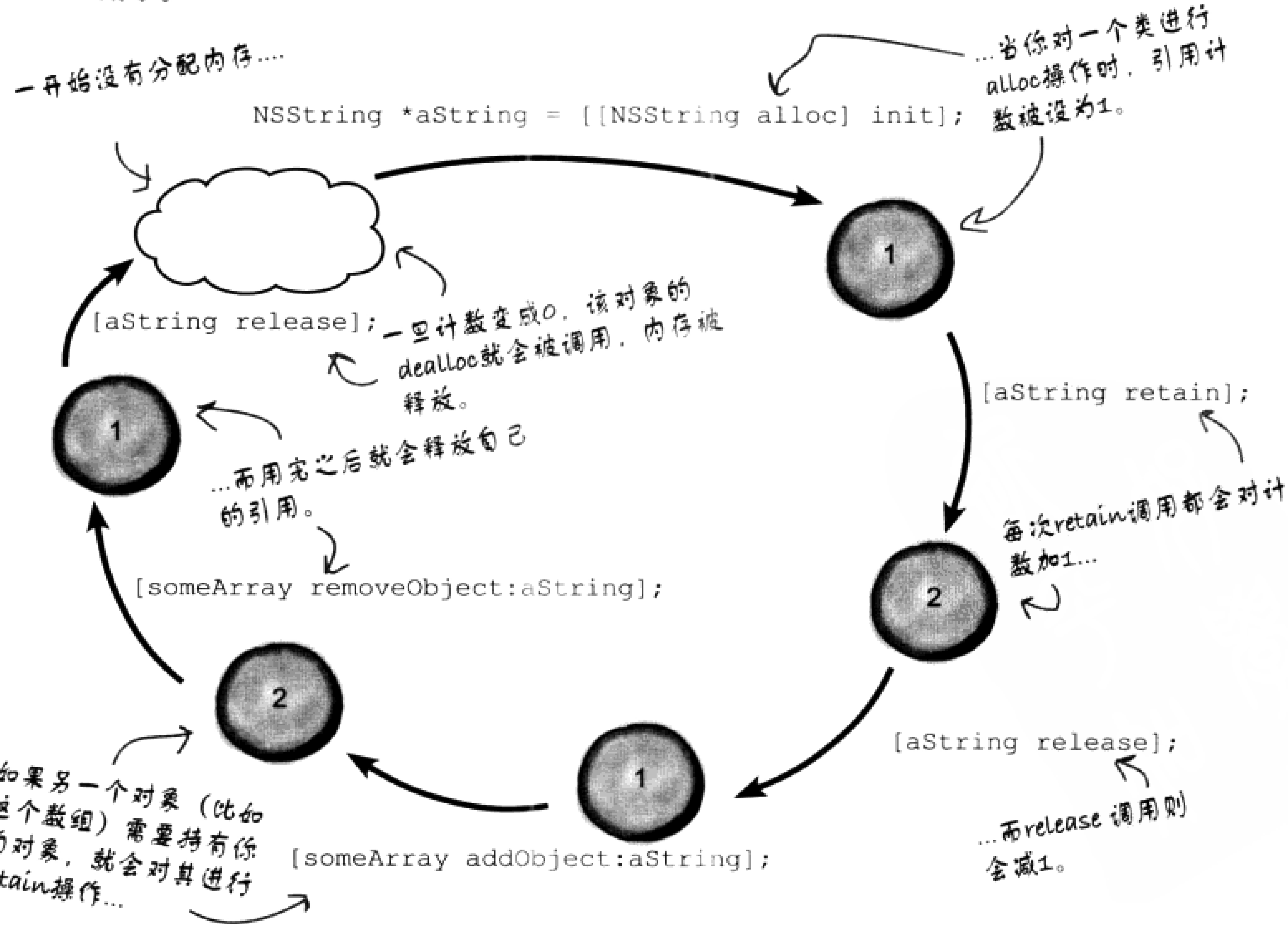
内存管理详解

```
- (void)dealloc {
        [tweetPicker release];
        [activities release];
        [feelings release];
    [super dealloc];
}
```

这些是你已经写过的一些内存管理的代码。

在iPhone上内存管理是绝对重要的，但这也不是说这件事很复杂。只要你坚守几条关键原则，就可以令应用井然有序，而不会出现那些可能会令你的应用被逐出App Store的内存泄露事故。

当你创建一个对象时，它的起始计数总是1，而你的不同操作可能会令计数增加或减少。当计数达到0的时候，对象就被真正释放，而对象占据的内存也就可以被再次利用了。



【拼吾爱-最新编程资源的分享下载站】

练习

请确认在下面的每段代码结束时，剩下的引用计数是多少，以及我们是否应该向每个字符串发送一个release消息。

最终引用计数

```
NSString *first = [[NSString alloc] init];
```

```
NSString *second = [[NSString alloc] init];
[someStringArray addObject:second];
```

```
NSString *third = [[NSString alloc] init];
[third autorelease];
```

```
NSString *fourth = [NSString
stringWithFormat:@"Do not read %@", @"Swimming
with your iPhone by TuMuch Monee"];
```

```
NSMutableArray *newArray = [[NSMutableArray alloc] init];
    NSString *fifth = [[NSString alloc]
initWithFormat:@"Read this instead: %@", "Financing your
iPhone 4G by Cerius Savar"];
    [newArray addObject:fifth];
    [newArray release];
```

```
NSString *sixth = [NSString stringWithString:@"Toughie"];
    NSArray *anotherArray = [NSArray
arrayWithObjects:sixth count:1];
    NSDictionary *newDictionary = [NSDictionary
dictionaryWithObjects:sixth    forKeys:@"Toughie"
count:1];
    NSString *ignoreMe = [sixth retain];
```

【拼吾爱-最新编程资源的分享下载站】



练 习

请确认在下面的每段代码结束时，剩下的引用计数是多少，以及我们是否应该向每个字符串发送一个release消息。

最终计数

```
NSString *first = [[NSString alloc] init];
```

1 引用计数将会是1，因为alloc自动将计数设为1。

```
NSString *second = [[NSString alloc] init];
[someStringArray addObject:second];
```

2 在执行完这段代码之后，变量“second”的引用计数将是2：1个来自alloc，1个来自被插入数组中。数组会自动持有加入其中的对象。

```
NSString *third = [[NSString alloc] init];
[third autorelease];
```

1 此处的引用计数仍然是由alloc设置的1，但是现在对象在自动回收池里面，也就是说在当前事件循环结束之后，会自动收到一个release。

```
NSString *fourth = [NSString
stringWithFormat:@"Do not read %@", @"Swimming
with your iPhone by TuMuch Monee"];
```

1 此处的引用计数为1，但是会被放到自动回收池中。这是由于我们不是通过alloc、new、copy或者mutableCopy这四种方法之一获得对象的。

【拼吾爱-最新编程资源的分享下载站】

请确认在下面的每段代码结束时，剩下的引用计数是多少，以及我们是否应该向每个字符串发送一个release消息。

最终计数

```
NSMutableArray *newArray = [[NSMutableArray
alloc] init];
    NSString *fifth = [[NSString alloc]
initWithFormat:@"Read this instead: %@",
"Financing your iPhone 4G by Cerius Savar"];
    [newArray addObject:fifth];
    [newArray release];
```

1

变量“fifth”的引用计数将会是1：

首先它会从alloc得到初始计数为1。

接着被插入到数组“newArray”中，导致其计数变为2。

然后因为数组被销毁时会对其中所有对象发送一次release，计数又变回1。

```
NSString *sixth = [NSString
stringWithString:@"Toughie"];
    NSArray *anotherArray = [NSArray
arrayWithObjects:sixth count:1];
    NSDictionary *newDictionary =
[NSDictionary dictionaryWithObjects:sixth
forKeys:@"Toughie" count:1];
    NSString *ignoreMe = [sixth retain];
```

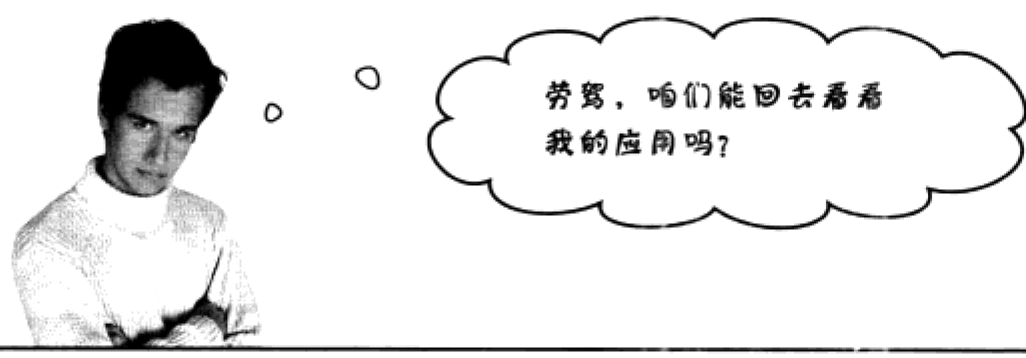
4

开始时，通过初始化方法创建（请注意并不是alloc，所以会被自动释放）的“sixth”引用计数为1并将被自动释放。

接着它被插入数组，又被持有一次。注意，数组也不是被alloc分配出来的，所以它也会被自动释放。

然后字典再一次持有它，字典也不是用alloc分配的，将被自动释放。

最后，一次显式的持有动作……
所以尽管“sixth”的引用计数为4，但我们开发者也仅需发送一次release消息给“sixth”，剩下的清理工作留给自动回收池就行了。



【拼吾爱-最新编程资源的分享下载站】

设计魔力贴答案（续）

使用你在为程序加入选取器和按钮时学到的知识，将下面的魔力贴与添加文本字段时要修改的方法或者文件对应起来。

1 在InstatwitViewController.h中 为UITextField添加一个IBOutlet和@property声明。

```
        IBOutlet        UIPickerView *tweetPicker;
    IBOutlet UITextField *notesField;
    NSArray* activities;
    NSArray* feelings;
}
@property (nonatomic, retain) UIPickerView* tweetPicker;
@property (nonatomic, retain) UITextField* notesField;
```

InstatwitViewController.h

将notesField加入@synthesize的行列

2 在InstatwitViewController.m的顶部 。

```
@synthesize tweetPicker, notesField;
```

InstatwitViewController.m

这里根据@property声明来合成访问方法。你可以加一行新的@synthesize语句，也可以把内容加在现有的行末尾，用逗号分隔。

3 在InstatwitViewController.m的dealloc方法中，添加[notesfield release]。

```
- (void)dealloc {
        [tweetPicker release];
        [activities release];
        [feelings release];
        [notesField release];
    [super dealloc];
```

InstatwitViewController.m

属性会自动持有传递给它的引用—我们需要在dealloc中释放它。

在我们的对象被释放时，dealloc方法会被调用，我们需要在这里释放对文本字段的引用。

对于便条字段，我们不需要动作方法或数据源

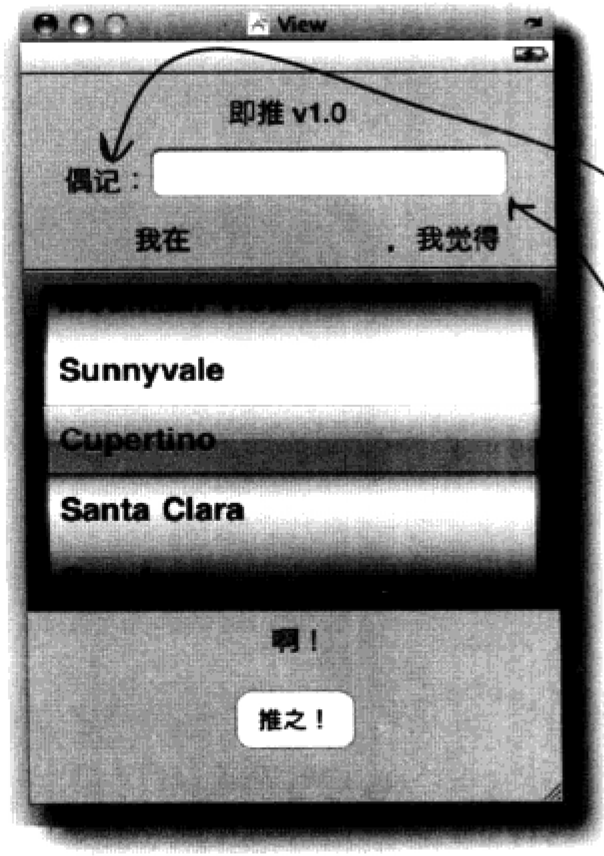
为UITextField添加一个IBAction

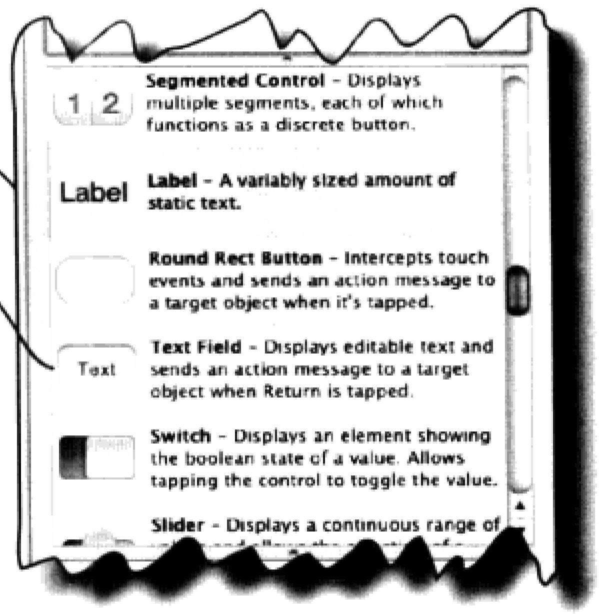
为notesField创建代理和数据源

4 使用Interface Builder，**在视图中加入一个UITextField**。

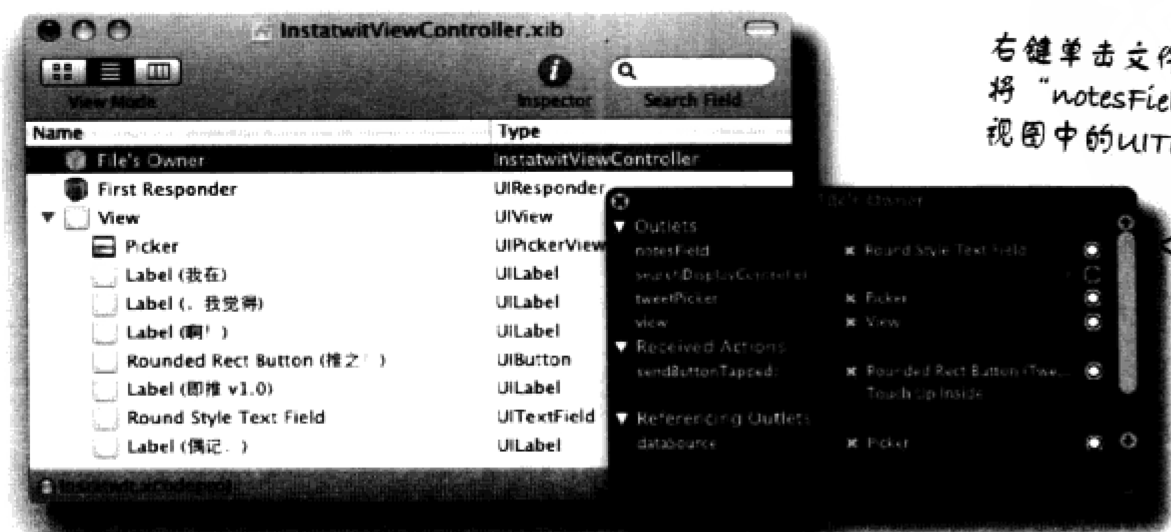
要做到这一点，你得打开InstatwitViewController.xib，并在控件库中找到文本字段。然后将文本字段拖曳到写着“即推 v1.0”和“我在……我觉得……”的两个标签中间。你还需要在文本字段前面放上一个标签，写上“偶记：”。

很可能你需要挪动选取器、标签和按钮的位置。





5 对于第1步中创建的属性，用Interface Builder，**把UITextField与IBOutlet连接起来**。



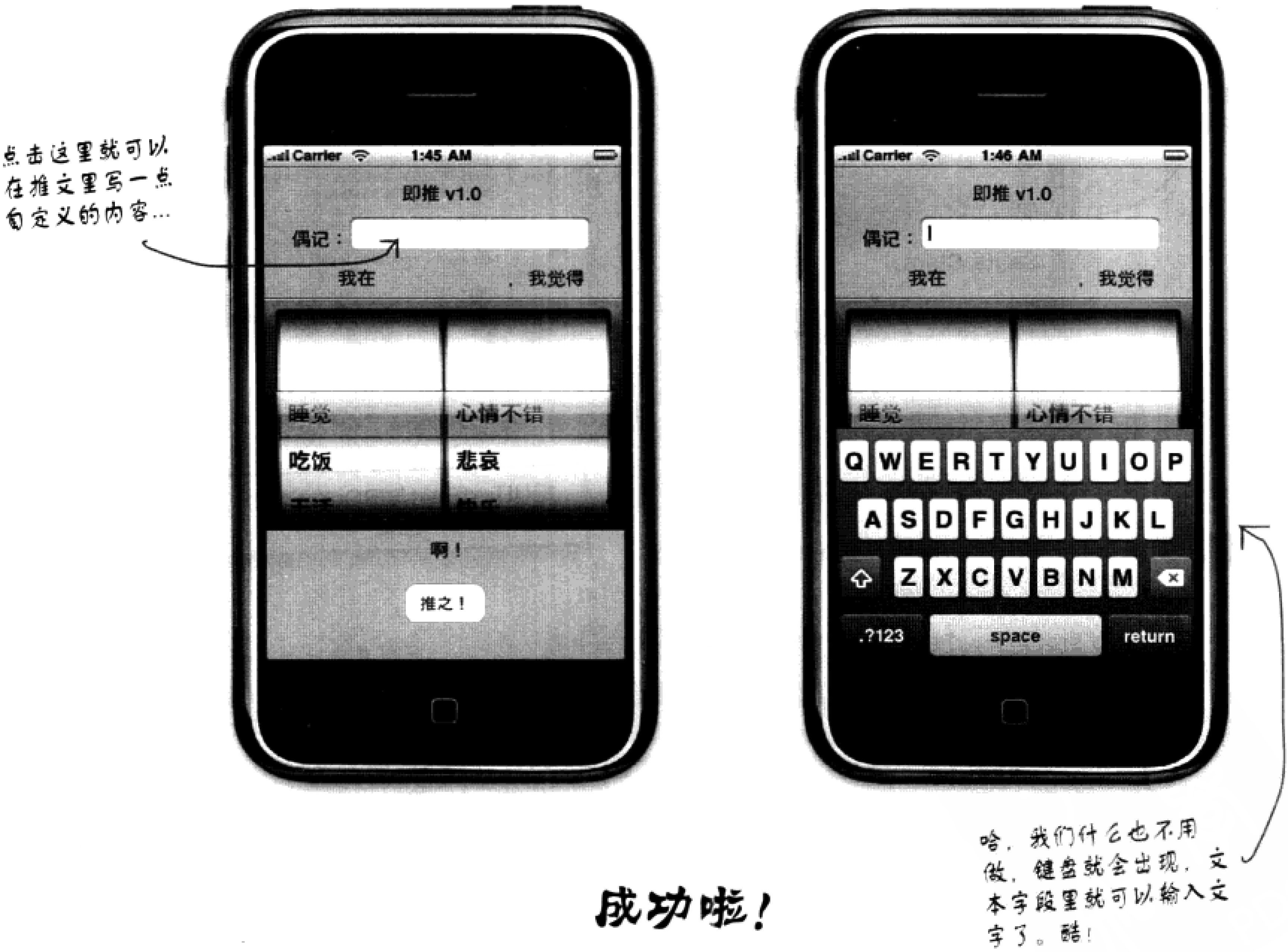
右键单击文件所有者，将“notesField”旁边的圆圈与视图中的UITextField连接起来。

将其保存，然后……

【拼吾爱-最新编程资源的分享下载站】

运行测试

现在已经保存好了一切，让我们回到Xcode中，编译执行应用，并启动模拟器。



【拼吾爱-最新编程资源的分享下载站】



Objective-C揭密

本周访谈：
你到底是谁？

Head First: 你好，Objective-C！感谢你接受我们的访问。

Objective-C: 谢谢！很高兴来到这里。最近由于iPhone这家伙，我受到不少关注。

Head First: 听说你的血统相当高贵，对吗？能不能给大家讲一讲你自己的情况？

Objective-C: 当然可以。我是一门面向对象语言，所以我拥有类和对象，但是我的根子有很浓厚的C背景。我的面向对象理念来自Smalltalk。说真的，我没有太多好说的。

Head First: 你说你来源自C背景，指的是什么？

Objective-C: 哦，几乎我的全部语法都和C的语法相似，比如循环、类型、指针等。你可以很容易地通过我来使用其他的C语言库，比如SQLite。诸如此类的事情。

Head First: 但你可不止这么点本领，是吧？

Objective-C: 嗯，是啊，确实如此。最明显的就是，我是面向对象语言，所以类啊，抽象接口（我管这叫协议），继承，等等，都工作良好。

Head First: 那内存管理怎么样？也像C那样用malloc和free吗？

Objective-C: 哦，malloc和free仍然可以工作，就像在C里面一样，但是对于对象，我还有一套非常好用的内存管理模型。我使用引用计数。

Head First: 啊——那么你会跟踪记录是谁在使用哪些对象？

Objective-C: 正是。如果你想要保留一个对象一段时间，只要对我说一声你要持有它的一个引用就行。用完了？释放你的引用就行了。当没有任何引用剩下时，我就会清理这个对象，把它占用的内存释放归还给你。

Head First: 真不错。还有其他花样吗？

Objective-C: 啊，当然了。你知道那些读写方法吧？就是在其他面向对象语言里，你需要编写的、用来包装类中成员变量的方法。我这里不用自己写。我可以自动为你生成这些方法。不仅如此，你还可以告诉我希望怎么来管理与之相关的内存。哦，还有一点是我的最爱：我可以无缝地向类中加入新的方法。这被称为类别。

Head First: 哦，这真有意思。我们快要到时间了，那么我问最后一个问题。那些到处都有的“NS”，到底是怎么回事？比如NSString和NSInteger。

Objective-C: 啊——那些都是CocoaTouch框架的组成部分。我之前提到过我的名门血统；在iPhone上用到的核心库，大部分都是来自CocoaTouch，这是来自OpenStep的Cocoa的一个移植版本，而后者来自NeXTStep，这也就是NS字样的来源。这些框架是用Objective-C编写的，但他们是框架，而不是语言的成分。当你编写iPhone程序时，你会一直使用像这样的一些东西。举例来说，字符串并不会用char *，通常你会用NSString或者NSMutableString。我们总是混在一起。

Head First: 这真是让我们大开眼界！再次感谢你来到我们的节目，祝你和iPhone一切顺利！

Objective-C: 不客气。感谢对我的邀请。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：如果我以后要用到一个对象，却又没持有它，会发生什么情况？

答：很有可能该对象的引用计数会变为0，并在你开始使用它之前被清除。这样你的应用程序就会崩溃。不过更令人头疼的是：在模拟器上程序也可能不会每次都崩溃。模拟器能使用的内存比实际的iPhone或者iPod Touch要多得多，因此行为也不同。所以应用可能会显得一切正常，直到你把它放到电话上测试为止。然后悲剧就发生了。

问：如果我把对象释放太多次会怎么样？

答：基本上和上面的情况一样。当引用计数达到0后，对象就会被释放而内存被回收。向已经回收的内存再次发送release消息，几乎肯定会令你的应用程序崩溃。

问：如果我的项目在模拟器上工作，在实机上就崩溃，是怎么回事？会是内存使用的问题吗？

答：绝对是。真实设备上的内存要比模拟器上紧张得多。我们在后面的章节里会对如何调试这些问题做进一步介绍，也会讲解如何用Instruments来跟踪内存使用与泄露的情况。

问：我怎么才能确认是否对内存进行了高效的管理呢？

答：iPhone SDK自带了一个非常好的内存工具，叫做Instruments。它可以显示出内存的使用方式、内存占用峰值、分配与回收内存的速度，以及最重要的一点，是否存在内存泄露。稍后我们会在书中详细介绍这一工具。

问：如果我把一个东西设置为nil会发生什么情况？

答：哦，这取决于你把什么东西设置成了nil。如果是本地变量，不会发生任何事。变量会变成nil，但是它指向的对象占用的内存仍被占用，所以几乎百分之百地会泄露一些内存。不过，如果是属性的话……

问：我必须持有要设置给属性的东西吗？

答：不用！嗯，可能不用。这就是为什么要在@property声明中使用“retain”参数。如果你用了retain，属性就会在自身被设置或清除时，自动向传入的值发送持有和释放消息。不过，还是要注意在你的dealloc方法中清除属性。如果你的属性使用了retain参数，而它在你的对象被释放时仍然有取值，那么无论该属性的值是什么，都必须被释放。你可以在dealloc里显式地向成员变量发送release消息，也可以把属性设置成nil。还有一点要注意：对于属性的自动持有和释放功能，只有在通过“.”表示的时候才会起作用。如果你直接修改属性背后的成员变量，属性对此就无能为力了，也不能正确地进行retain和release。

问：Objective-C不是跟Java和.NET一样，有自动垃圾收集功能吗？

答：实际上，在Mac平台是有的。但是苹果公司没有在iPhone OS上提供垃圾收集功能，所以你需要回归到引用计数，使用retain和release。

问：那malloc和free怎么样了？我还能使用它们吗？

答：可以，但是不能用于对象类型。Malloc和free对于基本的内存块仍然工作良好，就像在C中一样，但是在对类进行实例化的时候要用alloc。

问：你每次在alloc后面放的init调用是怎么回事？

答：Objective-C与其他面向对象语言不同，没有构造函数。相反地，根据传统，你可以提供一个或者多个init方法。对于分配的任何类，都应该调用init方法，所以你几乎总是看到它们挨在一起，就像是[[SomeClass alloc] init]这样。

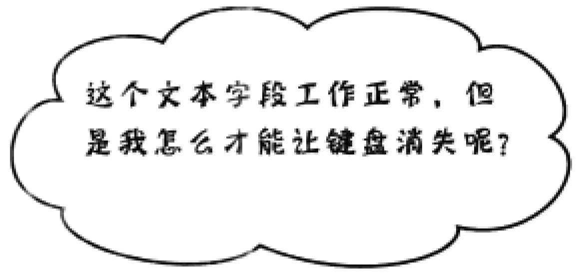
问：我怎么才能知道有没有别人持有我的对象呢？比如数组什么的？

答：基本上你用不着管它。要做的只是遵守内存使用规则，也就是说只要是用alloc、new、copy或者mutableCopy获取的对象，都必须向其发送release消息。对于其他情况，如果你以后还要用到这个对象，就对其进行配对的retain/release。除此之外你就不用管了，其他的类会处理好它们自己的内存管理问题。

问：我们不能把消息内容直接附在字符串的后面吗？

答：NSString是不可改变的，不过我们可以用NSMutableString。

但是当迈克输入完毕时……





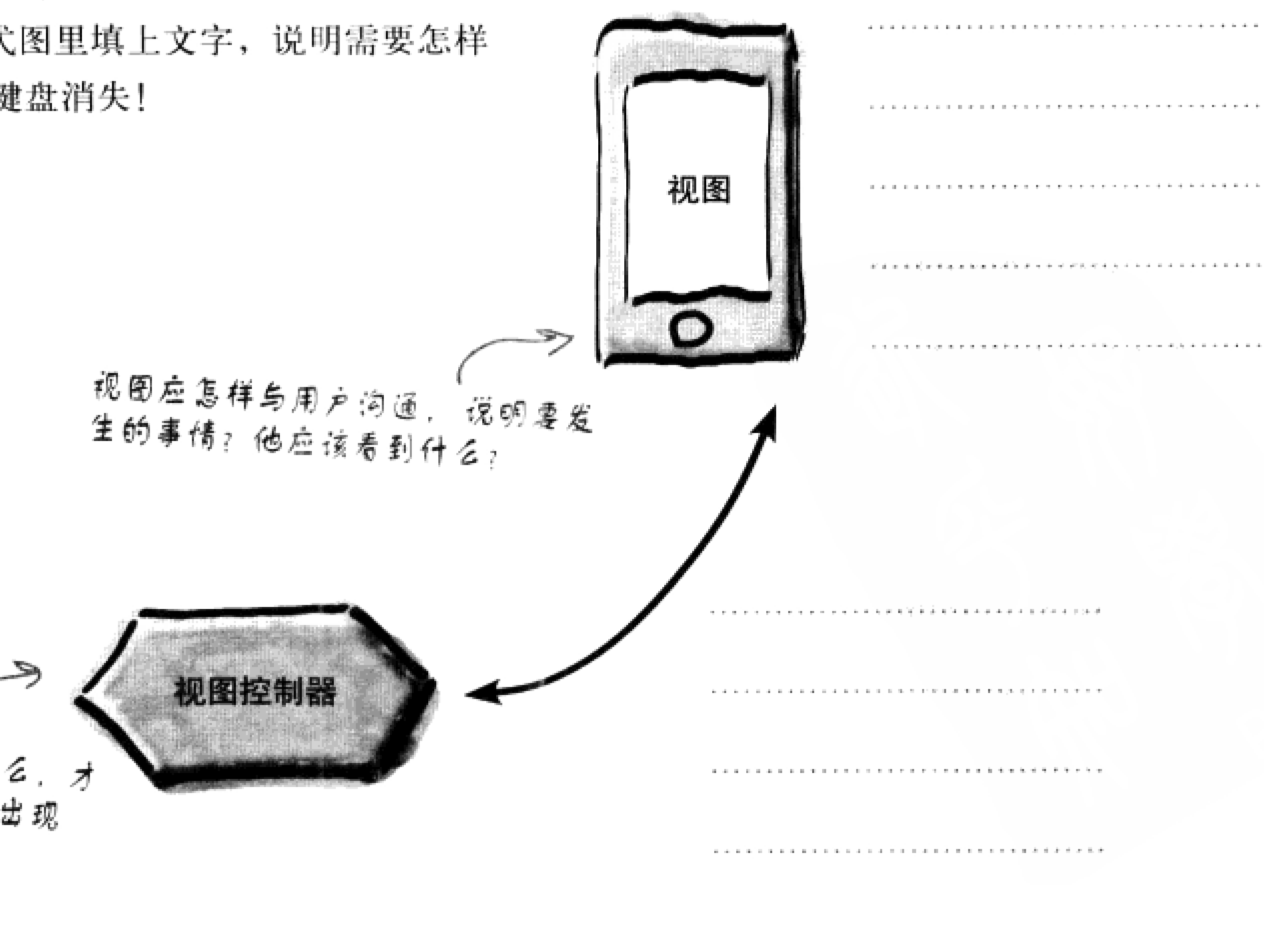
键盘是永存的吗？

行动起来，摆弄摆弄它，试着让键盘消失。回车键做不到这一点，点击屏幕上其他位置也没有效果。有点不太酷。

人人都是架构师



你的工作是作为架构师，规划键盘的行为。在下面的模式图里填上文字，说明需要怎样做，才能令键盘消失！

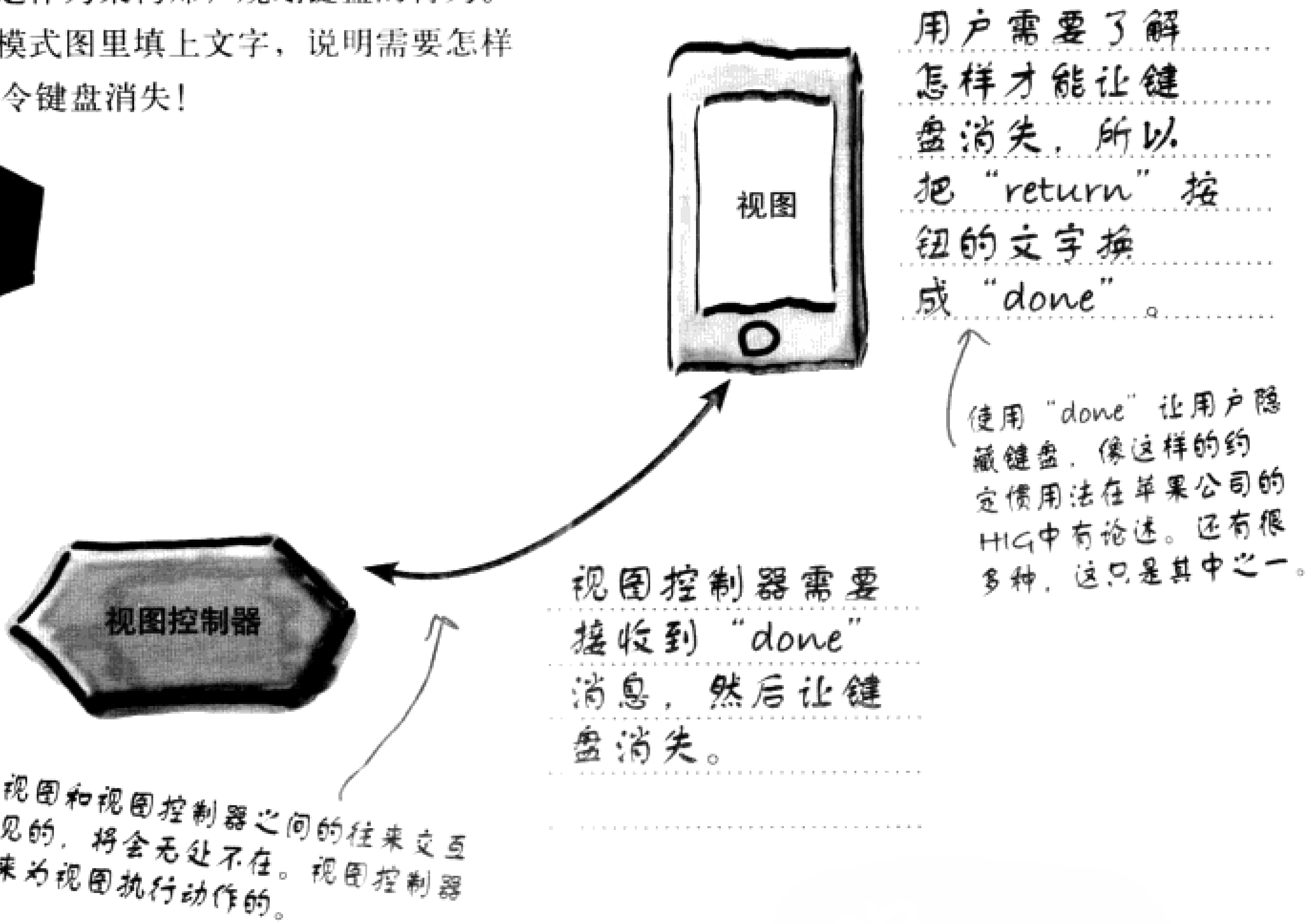


【拼吾爱-最新编程资源的分享下载站】

人人都是架构师—答案



你的工作是作为架构师，规划键盘的行为。在下面的模式图里填上文字，说明需要怎样做，才能令键盘消失！



有问必答

问： 一开始的时候，我们什么也没做，为什么键盘就会出现呢？

答： 当用户表明要与一个特定控件进行互动时，iPhone OS会将焦点交予该控件，并将其设置为事件的“第一响应者”。当某些控件成为第一响应者时，就会触发键盘自动显示出来。

让我们从视图开始。

【拼吾爱-最新编程资源的分享下载站】

定制你的UITextField

在Interface Builder中，选中迈克的定制字段并按⌘1，就会显示出查看器（Inspector），你可以在里面通过Text指定文本字段内容的初始值，Placeholder是字段中没有内容时显示的灰色文字，Alignment可以指定文字左右或者居中对齐，而Border可以定义不同的边框等。现在你还不用为字段添加任何内容，所以先让它们都空着。



接下来修改回车键的标签

想改变键盘中按钮的名字（这样就是“done”，而不是“return”了），需要使用查看器中的另一个选项。改变按钮标签的关键，是令其更清楚地与用户沟通，令他们明白怎么做才能让键盘消失。

单击Return Key下拉菜单，并选择Done。

默认情况下，Clear When Editing Begins是被选中的。这意味着只要用户点击了文本字段，不管之前里面有什么内容都会被清除。也就是说用户无法编辑内容。

对于“return”按钮的其他选择——其中有一些很直白（比如“Google”），其他的则要微妙一些。请查看苹果公司的HIG，了解该在何时使用其他那些值。



留神！

留神HIG！

除了Text和Placeholder域之外，对其他选项做修改有可能会降低应用的可用性，从而会令苹果公司不高兴，所以要小心一点。

现在，让键盘与视图控制器开始通话……

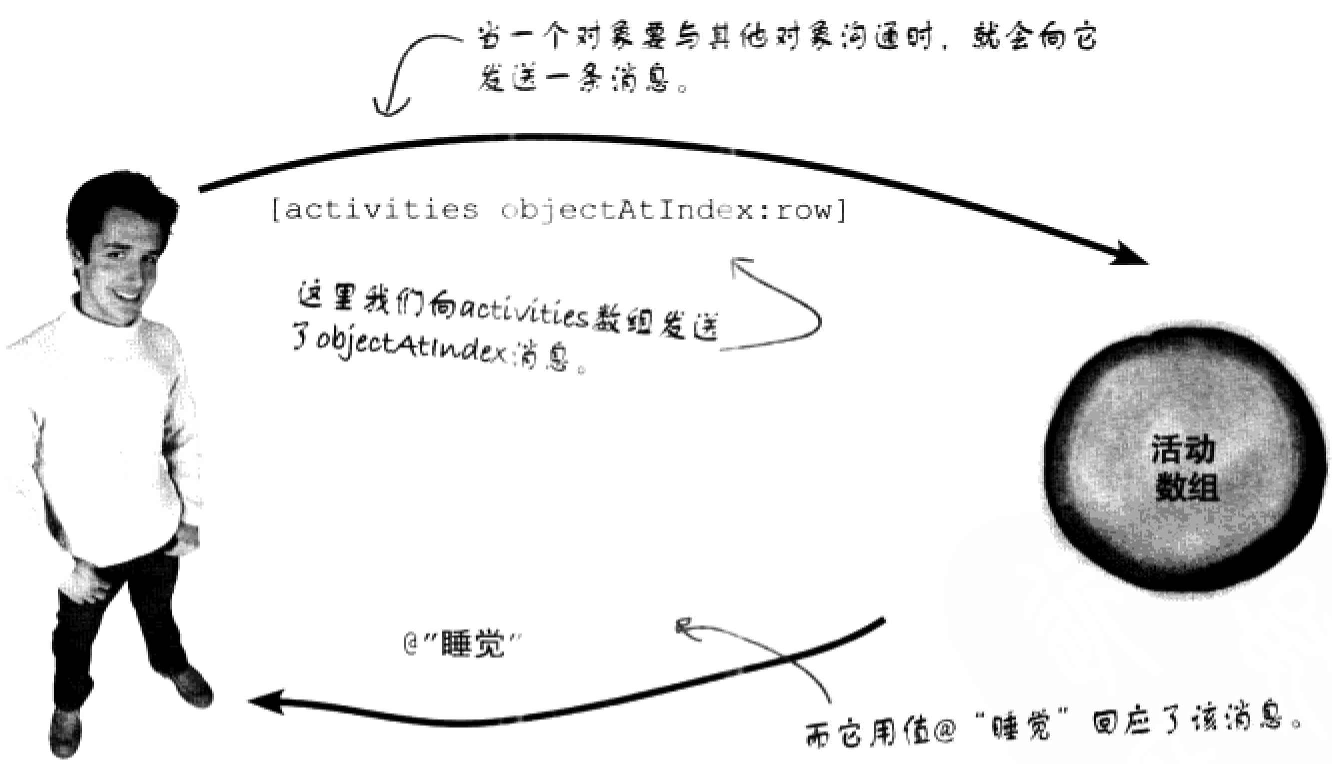
【拼吾爱-最新编程资源的分享下载站】

使用键盘的组件会请求它出现……

当用户点击文本字段时，iPhone OS会将焦点交予此控件，并将其作为此点击事件的“第一响应者”。有许多方法会让组件获得焦点：用户显式地点击控件；将键盘回车键的功能设置成移动到下一个待填充的控件；应用程序根据某些事件，显式地将某些控件设置为第一响应者等。当组件成为第一响应者时，其行为也因“人”而异，但是对于文本字段来说，其行为就是请求iPhone OS显示键盘。组件与应用程序之间进行的这些交流，是编写一个应用程序的基础，在消息传递的过程中随处可见。

……方法是向其他对象传递消息

其中的理念是，无论何时只要一个对象（不管这个对象是你的应用代理，还是另一个组件，或者是iPhone中的GPS）想让其他对象做些事情，都要向它发送一条消息。



在Objective-C中，你向一个对象发送消息，而它则对该消息做出回应（相当于一个方法的返回值）。Objective-C运行库会将这些消息转换成对象或者类（如果是静态方法）的方法调用，但是请习惯于将这些当作消息来对待；在苹果公司的文档里，你会经常见到这样的话：“此消息的接收者将会……”。现在，让我们在用户用完键盘的时候，用消息传递来打发走它。

【拼吾爱-最新编程资源的分享下载站】

让文本字段让出焦点

为了让键盘消失，我们需要告知文本字段，用户已经使用完毕。我们可以通过请求文本字段退出第一响应者的状态来做到这一点。

在Objective-C中发送消息是很轻松的：只要依次列出消息的接收者、要发送的消息以及需要传递的所有参数，就可以了。

```
[notesField resignFirstResponder];
```

将消息用方括号括起来。

这是消息的接收者——对我们来说，就是notesField。

这个位置就是真正的消息。对于我们来说不需要任何参数，所以这些就足够了。要了解每个组件到底都能对哪些消息做响应，可以查看苹果公司的文档。

这是一条语句，与其他语句一样——所以别忘了分号。



这不就是视图给视图控制器发送信息的方法吗？

正是！我们的视图控制器可以对许多消息做出响应，比如sendButtonTapped和viewDidLoad。

一直以来，你都在对消息做出响应。下面就是技巧所在：当用户点击键盘上的完成按钮时，文本字段可以发送一条消息。我们要做的只是告诉它，我们的视图控制器对于正在发生的事情感兴趣。



你可以向nil发送消息，从表面上看起来没有任何问题。

Objective-C允许你向nil发送消息，而不会有任何警告。如果你习惯于其他语言中的NullPointerException之类的错误，这可能会令调试更麻烦。在进行调试的时候，要小心未初始化的变量或者作为其他类型返回的nil值。

【拼吾爱-最新编程资源的分享下载站】

消息处理详解

从第1章开始，你就一直在处理消息了，但是我们并没有好好地解释这一切背后的语法，而仅仅是把方法声明放在头文件中，将实现放在.m文件里。下面是一些片断，摘自InstaTwit中我们对sendButtonTapped的实现。

实现文件（.m）的开头就是@implementation，然后是要实现的类的名称。

```
@implementation InstatwitViewController
```

在消息名称前面的括号里指定响应的类型。

接着是消息名称。完整的消息名称（包括参数）一般会很长并且描述详尽，在Obj-C中被称作一个选择子。

如果你的消息有参数，则在消息名后加一个冒号，然后是本地变量的类型和名称。额外的参数也会有名称和类型，以及变量名。

```
-(IBAction)     sendButtonTapped: (id) sender {
    新方法的代码放在这里。
}
```

最终，得到了我们在接口中声明的方法的实现。

减号“-”表示这是一个实例方法。（而加号“+”表示静态或类方法）。

在头文件中声明消息的语法，与在实现文件中的语法一样。

```
@end
```

实现文件以@end结束。

InstatwitViewController.m

【拼吾爱-最新编程资源的分享下载站】

Objective-C中的消息使用有名称的参数

Objective-C中，消息名称一般都很长，而且对消息描述详尽。这在你使用附加参数的时候会变得很有意义。当你发送一个带有参数的消息时，消息和参数的名称都是特定的。Objective-C的消息读起来更像一句话。让我们看一看UIPickerViewDataSource中的方法声明。这个方法返回一个选取器视图中给定部件的行数。其声明如下：

返回类型 / 方法名称 / 第一个参数的类型 / 本地参数名称 / 第二个参数的公开名称

```
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfR
owsInComponent:(NSInteger)component;
```

第二个参数的类型 / 第二个参数的本地名称

方法的参数可以有内部和外部名称；外部名称用来发送消息给接收者。这样当iPhoneOS要将这条消息发送给我们的代理时，它创建的调用应该类似于下例：

接收者 / 消息名称 / 值 / 第二个参数的名称 / 第二个参数的值

```
[pickerDelegate pickerView:somePicker numberOfRowsInComponent:component];
```

有问必答

问：你不停地在方法和消息这两个术语之间换来换去。哪个才对？

答： 两个都对，这完全取决于上下文。在Objective-C中，你向对象发送消息，而对方做出响应。Objective-C的运行库会把你的消息转换成一个方法调用，调用会有返回值。所以一般我们会说向某个接收者发送消息，但是如果你是在实现响应过程，那就是在实现一个方法。

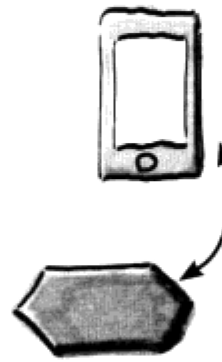
问：那么关于方法的那些参数……冒号前面的名字和类型后面的名字是什么关系？

答： 在Objective-C中，你可以为参数取一个公开名称和一个本地名称。当有人想要向你的对象发送该消息的时候，公开名称就会被用作选择子的一部分。这就是冒号前面的名字。类型后面的名字是本地名称；这是持有参数值的变量的名字。在Objective-C中，这两个名称不必相同，所以你可以为使用你的类的人准备一个明白易懂的公开名称，而在自己的代码里使用一个简便的本地名称。还有一件有趣的事：公开名称是可选的。如果你不提供公开名称，其他人在向你的对象发送该消息时，直接使用冒号和参数值就行了。很显然，参数的顺序是很关键的。

马上我们就会进一步讲解选择子。

【拼吾爱-最新编程资源的分享下载站】

用消息传递告诉视图控制器完成按钮何时被按下



此处消息在UITextField和控制器之间传递。

文本字段可以告知我们的视图控制器，键盘上的完成按钮何时被按下；我们只需要让它知道发送哪个消息就可以了。我们可以用Interface Builder来做到这一点。首先你得在.h文件里声明一个动作，然后在.m文件里实现它：

1 向InstatwitViewController.h里加入 IBAction。

就像我们对“推之”按钮那样，回到Xcode中，加入下面的代码：

```
- (IBAction) sendButtonTapped: (id) sender;
- (IBAction) textFieldDoneEditing:(id) sender;
@end
```

InstatwitViewController.h

这个方法原型（返回类型为void，有一个类型为‘id’的参数）是IBAction所必需的。

注意到了吗？IBAction被括起来了。它就是返回类型——不过被偷偷用#define定义成了void。

这是新的动作方法。前面的减号说明它是实例方法，名字叫textFieldDoneEditing，接受一个参数。参数sender的类型是id（指向一个对象的指针）。

2 在InstatwitViewController.m里添加方法的实现。

现在我们有了一个动作方法，会在完成按钮按下时被调用，我们所要做的就是让文本字段退出第一响应者状态，这样就会隐藏键盘。

```
//      END TWITTER BLACK MAGIC
}
- (IBAction)textFieldDoneEditing:(id) sender
{
    [sender resignFirstResponder];
}
```

InstatwitViewController.m

由于sender就是那个UITextField，我们可以直接把resignFirstResponder发送给它。

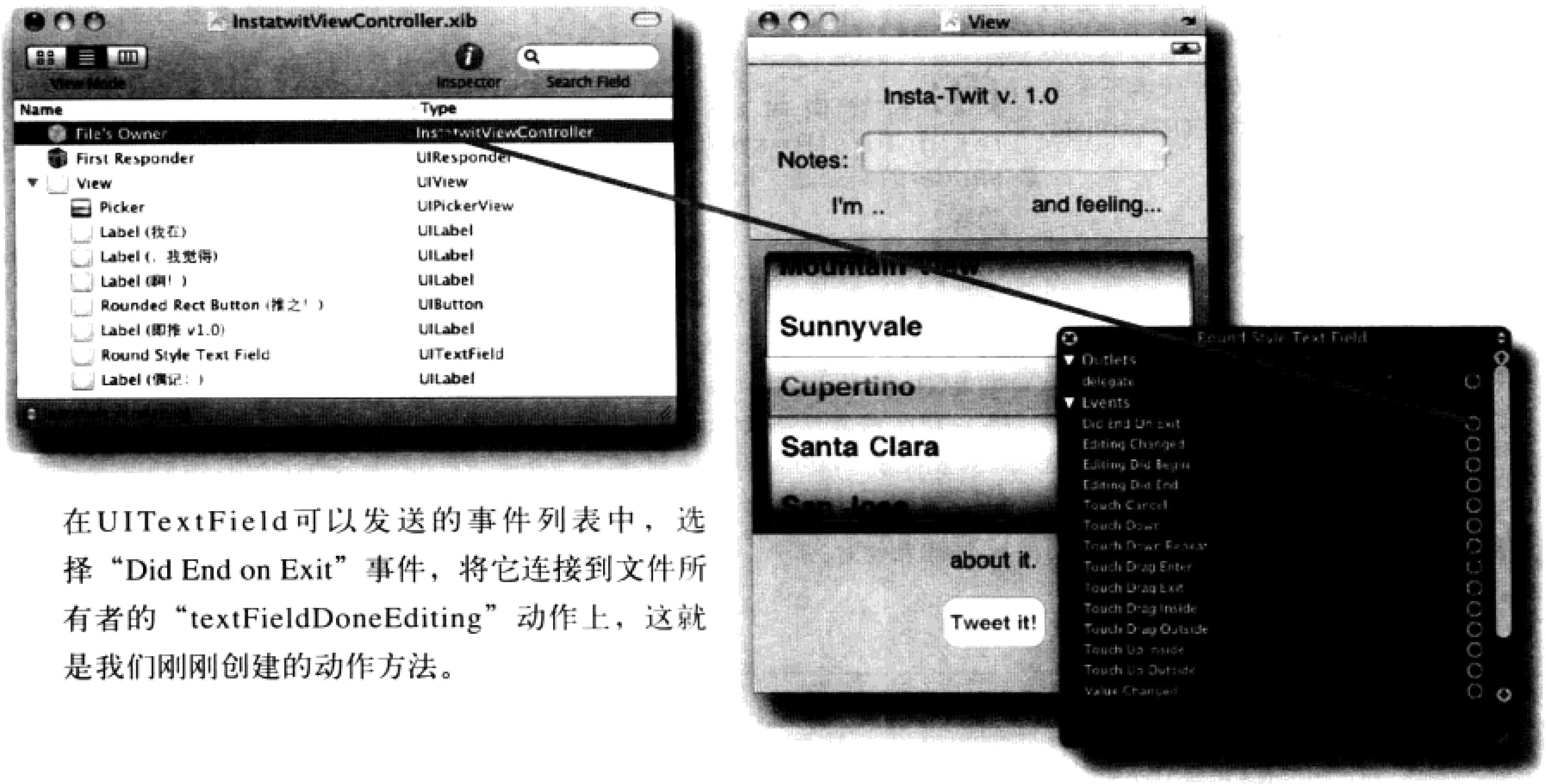
参数sender就是触发这个事件的组件。对于我们来说，就是那个UITextField。

差不多了，我们只需把它连起来……

【拼吾爱-最新编程资源的分享下载站】

❸ 在Interface Builder中与UITextField的事件联系起来

现在动作方法已经声明好了，双击InstatwitViewController.xib，回到Interface Builder中。右键单击UITextField，你就可以打开连接窗口。



在UITextField可以发送的事件列表中，选择“Did End on Exit”事件，将它连接到文件所有者的“textFieldDoneEditing”动作上，这就是我们刚刚创建的动作方法。

极客点滴

与圆角矩形按钮一样，UITextField也可以发起许多事件。请花一点时间，仔细看看出现的事件列表。通过查看器可以对字段做各种各样的定制，你可以把不同的事件（甚至多个事件）连接到不同的响应方法，从而与字段进行互动。请记住这一点，在你的应用中也许用得到。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：为什么在我们的动作方法里，要把消息发回给sender，而不是直接给notesField属性呢？

答： 哪种方法都没问题；因为它们都是指向同一个对象的引用。我们使用参数sender，是因为这样一来不管我们是否有一个属性引用了 UITextField，代码都可以正常工作。

问：你们提到了选择子，但我还是弄不清楚这是什么。

答： 选择子是方法的唯一名，当Objective-C将一条消息翻译成一个实际的方法调用时，就要用到这个名字。它基本上就是方法名加上用冒号分开的各个参数名。比如说，66页中的代码用到了一个选择子pickerView:numberOfRowsInComponent。在后面的章节里，你还会见到它们出现，尤其是当我们在代码中进行界面连接的时候。目前Interface Builder会替我们处理这些事。

问：当我们向sender发送resignFirstResponder消息时，sender的类型是“id”。这怎么能成功呢？

答： “id”是一种Objective-C类型，可以指向任何Objective-C对象。它就像是C++里的void*。由于Objective-C是一种动态类型语言，它完全可以向一个类型是“id”的对象发送消息。该对象是否可以响应这个消息，将会在运行时来确定。

问：如果对象不能响应这个消息，会怎样呢？

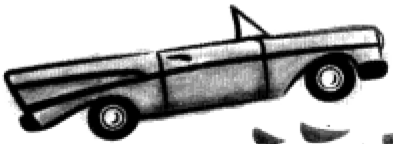
答： 你会得到一个异常。正是由于这个原因，所以只要有可能，你就应该使用强类型变量。不过，有些时候通用类型会更适合一些，比如在回调方法中，sender可以是许多不同种类的对象。

问：那说真的，方括号就是用来传递消息的吗？

答： 是的。而且也会用在数组下标索引中。我们都必须习惯这一点。

关键要点

- 在Objective-C中，你会向接收者发送消息。运行库会把这些消息映射成方法调用。
- 方法声明在头文件(.h)中进行，位于接口的结束大括号之后。
- 方法实现在实现文件(.m)中进行，位于@implementation和@end之间。
- 方法的参数通常都是有名称的，这些名称在发送消息时会用到。
- 参数可以有一个内部名称和一个外部名称。
- 用减号“-”表示实例方法；加号“+”表示静态方法。



运行测试



输入些内容，继续！

成功了！键盘消失了，现在你可以随便摆弄文本字段，添加一些文字了。

请记住，如果你在代码里加进了你的账户信息，那每次发推都会实际发送上去！

点击"完成"按钮...

【拼吾爱-最新编程资源的分享下载站】

还是有些不对劲

迈克准备试用这个自定义字段，看看会发生什么，但是当他输入了自定义消息之后……





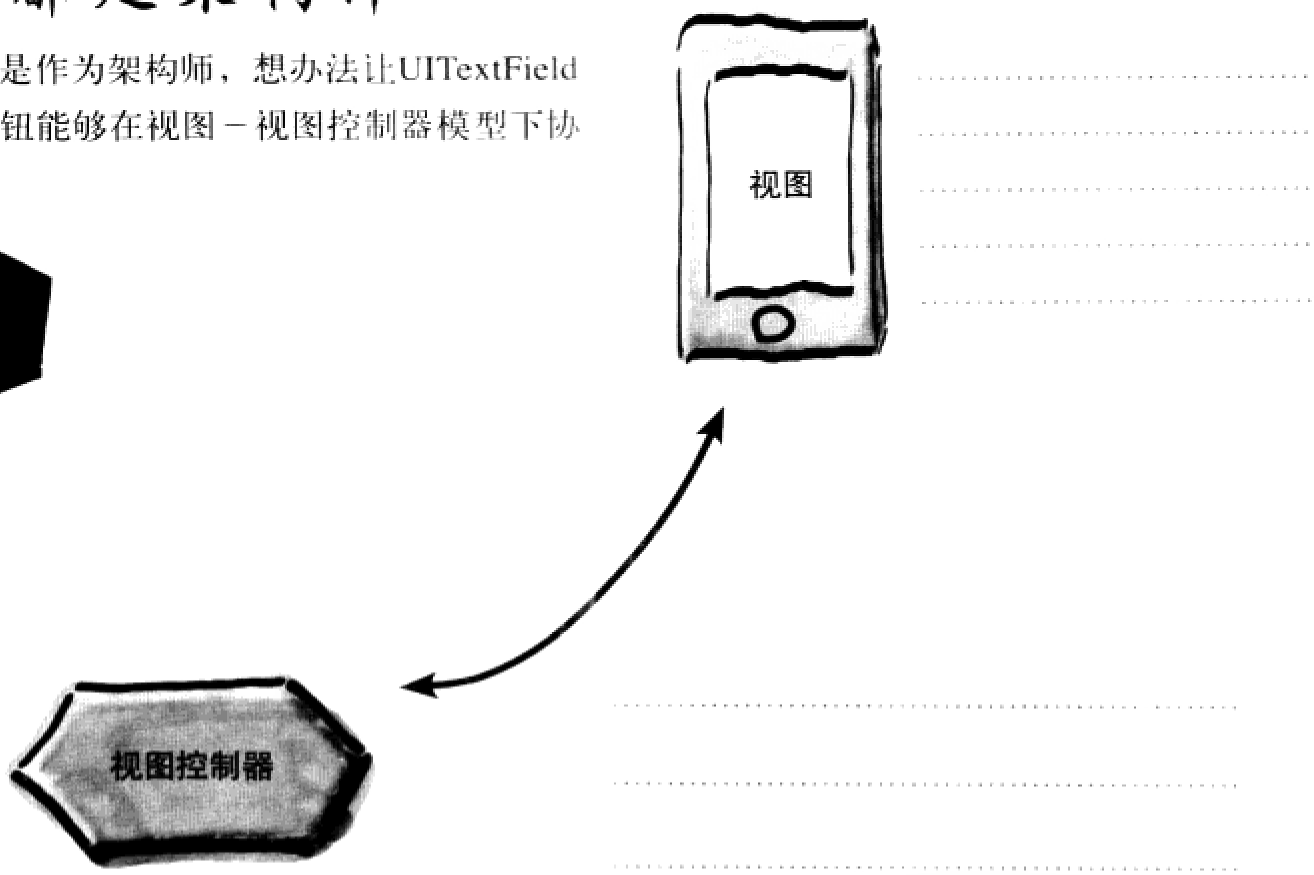
你现在已经掌握了事件和消息传递，要改正这个错误一点问题也没有……

【拼吾爱-最新编程资源的分享下载站】

人人都是架构师



你的工作是作为架构师，想办法让UITextField和发推按钮能够在视图－视图控制器模型下协同工作。

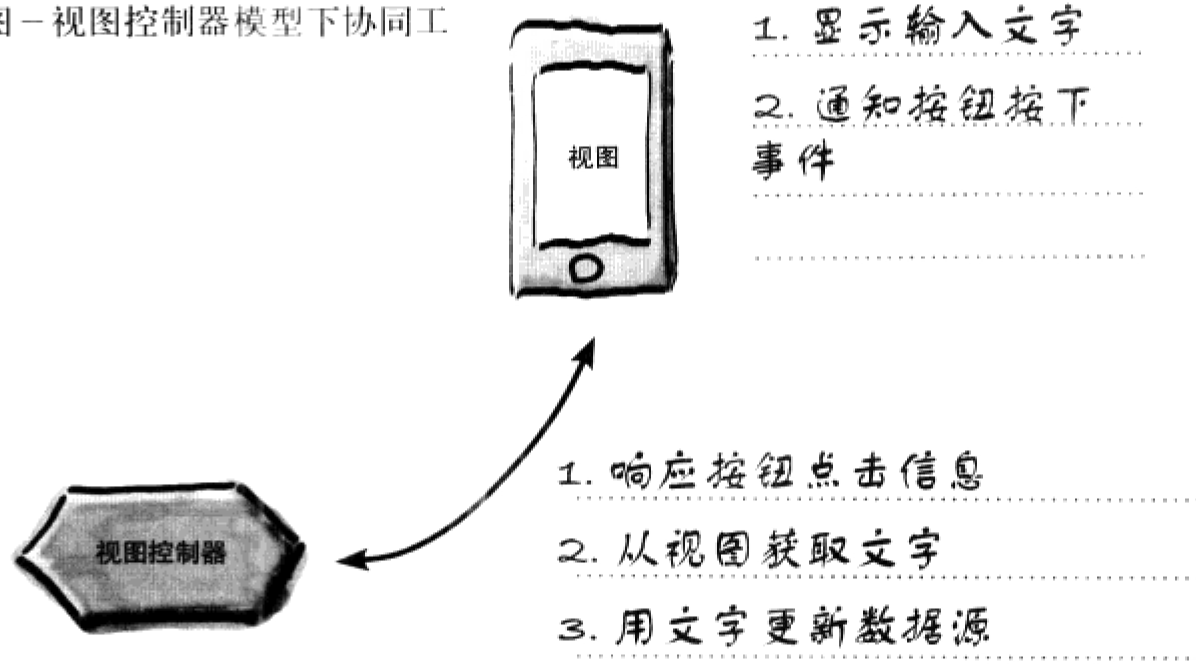


【拼吾爱-最新编程资源的分享下载站】

人人都是架构师一答案

你的工作是作为架构师，想办法让UITextField和发推按钮能够在视图－视图控制器模型下协同工作。





用字符串打造推文

我们需要把输入的文字集成到我们的推文中。为此，我们打算使用核心字符串类对字符串做一点操作。你以前已经构造过发往Twitter的消息，但是这次我们要加进去更多文字。在你对代码进行重构以便发送含有新内容的推文之前，让我们先仔细看一看你在第2章中的成果：

这个字符串并非来自alloc、new、copy或者mutableCopy，所以会被自动回收。

这是NSString的一个静态方法，接受一个字符串作为格式，并将格式串中的占位符替换成你在参数中提供的值。

引号前面的@表示这个字符串是一个NSString，而不是char*。

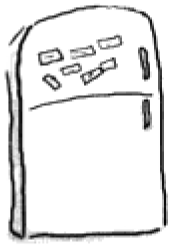
```
NSString* themessage = [NSString stringWithFormat:@"我在%@，我觉得%@啊！"
        [activities objectAtIndex:[tweetPicker selectedRowInComponent:0]],
        [feelings objectAtIndex:[tweetPicker selectedRowInComponent:1]]];
NSLog(themessage);
```

NSLog会把你传给它的任何字符串都打印到控制台。应用的最终用户不会看到这些信息。

这里我们用stringWithFormat来创建消息字符串。注意这个%@，这是一个为字符串准备的占位符。

现在你要做的，就是更新这些代码，将文本字段中的文字包括进来。看一看下一页的魔力贴，尝试让它们能够正常工作。

【拼吾爱-最新编程资源的分享下载站】



Xcode魔力贴

你需要修改InstatwitViewController.m文件，在消息中加入定制字段的内容。运用你刚刚学过的信息以及下面的魔力贴，将空缺的代码填入空格中。

```
- (IBAction) sendButtonTapped: (id) sender {
    NSString* themessage = [NSString stringWithFormat:@"  ____________
    我在%@，我觉得%@啊！",

    ________________  _____  __________________  __________  _____

    [activities objectAtIndex:[tweetPicker
selectedRowInComponent:0]],
    [feelings objectAtIndex:[tweetPicker selectedRowInComponent:1]]];
    NSLog(themessage);
```

```
[[NSString alloc] init];
```

```
,
```

```
:@""
```

```
notesField.text
```

```
%@.
```

```
[themessage release];
```

```
themessage
```

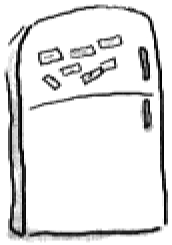
```
notesField.text
```

```
sender resignFirstResponder
```

```
?
```

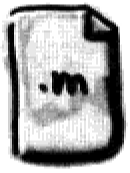
```
notesField.text
```

【拼吾爱-最新编程资源的分享下载站】



Xcode魔力贴答案

你需要修改InstatwitViewController.m文件，在消息中加入定制字段的内容。运用你刚刚学过的信息以及下面的魔力贴，将空缺的代码填入空格中。



InstatwitViewController.m

这是我们为自定义文字新增的占位符。

```
- (IBAction) sendButtonTapped: (id) sender {
    NSString* themessage = [NSString stringWithFormat:@" %@.
    我在%@，我觉得%@啊！",
    notesField.text  ?  notesField.text  :@""  ,
    [activities objectAtIndex:[tweetPicker
selectedRowInComponent:0]],
    [feelings objectAtIndex:[tweetPicker selectedRowInComponent:1]]];
    NSLog(themessage);
```

我们必须处理用户没有输入任何文字的情况。如果文本字段是空的，它的text属性将会是nil。在此我们使用了C风格的三元操作符。如果notesField.text不是nil，就会使用notesField.text里的值....

...而如果它的值是nil，我们会发送一个空字符串。要记住，这里必须是一个NSString，所以我们在引号前放了一个@。

这个问号（?）就是三元操作符，与Java和C++中一样，如果表达式为真则返回第一个值，否则返回第二个值。

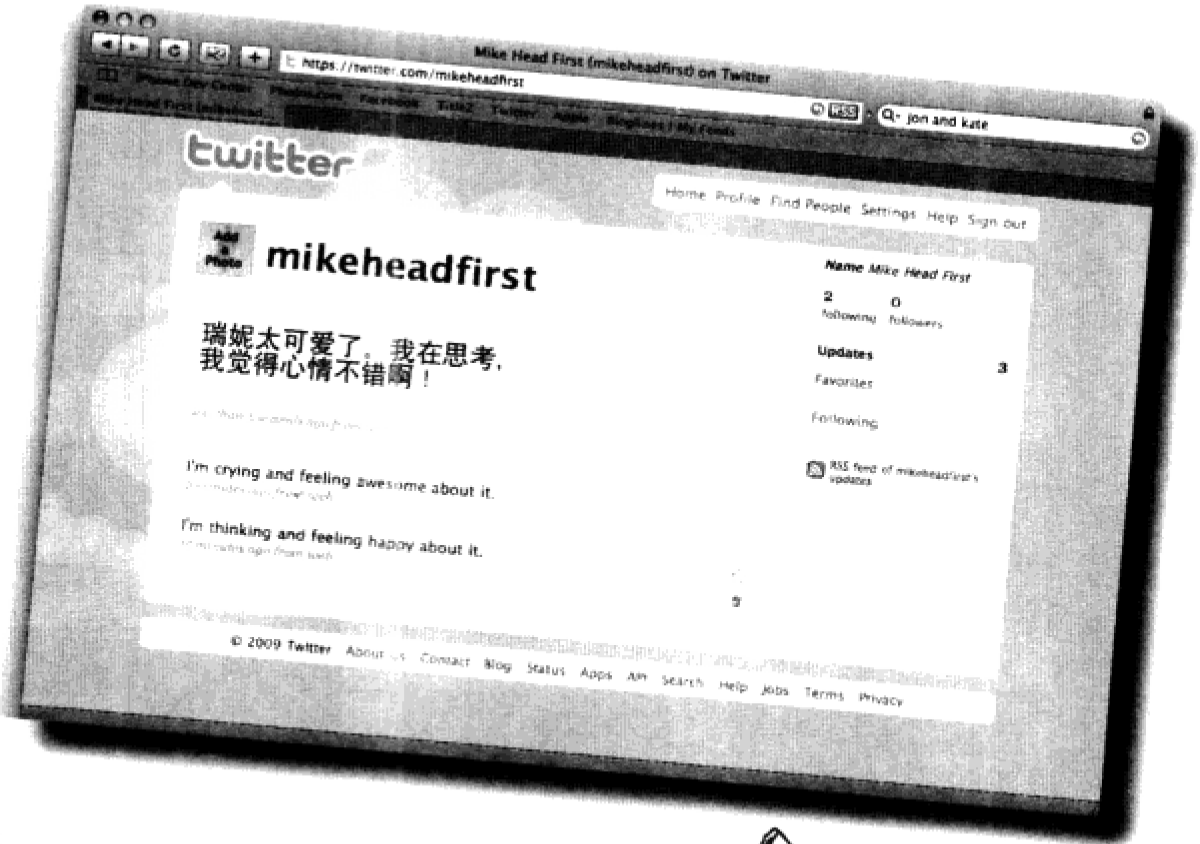
```
[[NSString alloc] init];
[themessage release];
themessage
sender resignFirstResponder
notesField.text
```

【拼吾爱-最新编程资源的分享下载站】

运行测试

继续前进，构建并运行应用，看看添加新文字的代码的效果。





现在里面有自定义的文字了！帅呆了！

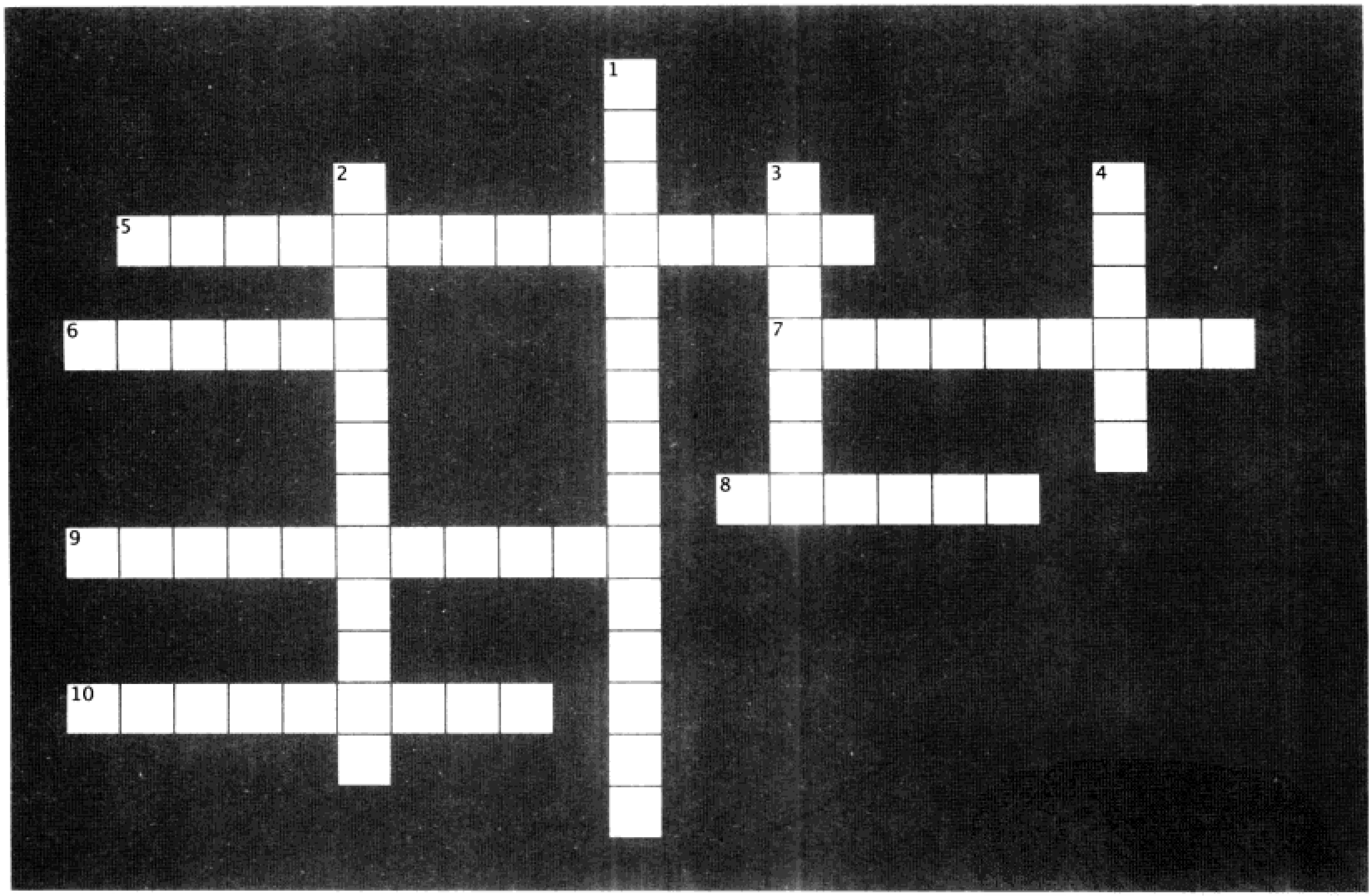


这可真是太棒了，我们可以相互交流感情了……

【拼吾爱-最新编程资源的分享下载站】

Objective-C纵横字谜

练习一下那些你新学的Objective-C术语。



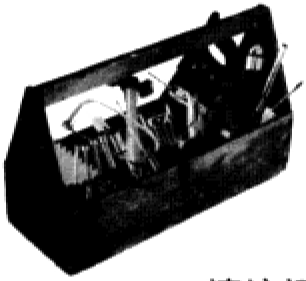
横向提示

5. 具有当前焦点的控件处于________状态。
6. 用它来集成另一个文件。
7. 经Objective-C翻译之后的方法的唯一名称，叫做________。
8. 通知编译器保有一个对象。
9. 自动方法。
10. 告诉编译器忽略互斥锁。

纵向提示

1. 一个对象数组，会在当前事件处理完之后被释放。
2. 在方法声明前面有一个“+”，表示该方法是________。
3. 在对象之间传递的东西。
4. ________管理对于iPhone应用非常重要。

【拼吾爱-最新编程资源的分享下载站】



你的Objective-C工具箱

你已经消化吸收了第3章，现在可以把Objective-C加到你的工具箱中。如果你想得到一份完整的书中功能提示的列表，请访问http://www.headfirstlabs.com/iphonedev。

特性	你会在...的时候用它
readwrite	当你希望属性可以被其他人修改的时候。编译器会分别生成读取和写入方法。这是默认值
readonly	当你不想让其他人修改属性的时候。你仍然可以改变属性背后的成员变量的值，但是编译器不会生成写入方法
assign	当你使用基本类型的时候，比如int、float等。编译器仅仅会创建一个简单的写入方法，只有一条myField = value这样的语句。这也是默认值，但是一般不是你期望的用法
retain	当你使用对象作为属性值的时候。编译器会持有传入的值（马上我们就会进一步解释持有操作），并且会在有新值传入的时候释放旧值
copy	When you want to hold onto a copy of some value instead of the value itself. For example, if you want to hold onto an array and don' t want people to be able to change its contents after they set it. This sends a copy message to the value passed in then retains that.

Objective-C

- 是编写iPhone应用的语言
- 是一种面向对象语言
- 具有先进的内存管理
- 使用消息传递与动态类型
- 具有继承和接口

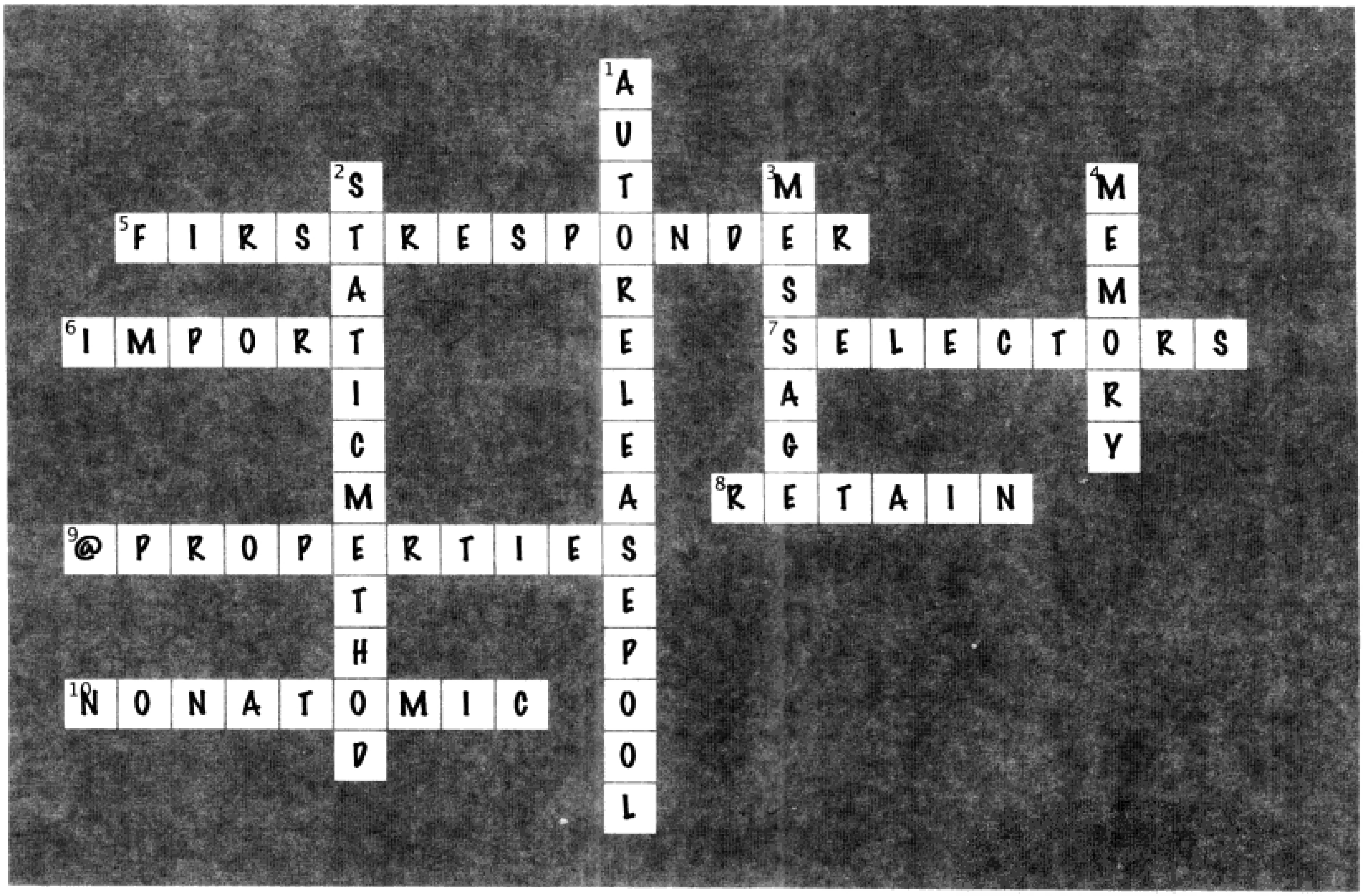
内存管理

- 你用alloc、new、copy和mutableCopy创建的对象必须由你自己释放
- 其他对象都可以当成是引用计数为1且位于自动回收池中

【拼吾爱-最新编程资源的分享下载站】

Objective-C纵横字谜答案

练习一下那些你新学的Objective-C术语。



横向提示

5. 具有当前焦点的控件处于__________状态。[FIRSTRESPONDER]

6. 用它来集成另一个文件。[IMPORT]

7. 经Objective-C翻译之后的方法的唯一名称，叫做__________。[SELECTORS]

8. 通知编译器保有一个对象。[RETAIN]

9. 自动方法。[@PROPERTIES]

10. 告诉编译器忽略互斥锁。[NONATOMIC]

纵向提示

1. 一个对象数组，会在当前事件处理完之后被释放。[AUTORELEASEPOOL]

2. 在方法声明前面有一个“+”，表示该方法是__________。[STATICMETHOD]

3. 在对象之间传递的东西。[MESSAGE]

4. __________管理对于iPhone应用非常重要。[MEMORY]

【拼吾爱-最新编程资源的分享下载站】

4 多视图

表格中的视图



大多数iPhone应用中的视图数目都不止一个。我们已经用单个视图编写了一个很酷的应用，但是用过iPhone的人都知道，大多数应用可不是像这样的。有一些更引人注目的iPhone应用使用多个视图，在处理复杂信息方面的工作卓有成效。我们打算先从导航控制器和表格视图开始，这两者就类似你在邮件应用与联系人应用所见到的那种界面。只不过我们会做一点小小的变化……

【拼吾爱-最新编程资源的分享下载站】

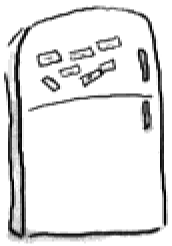




脑力举重

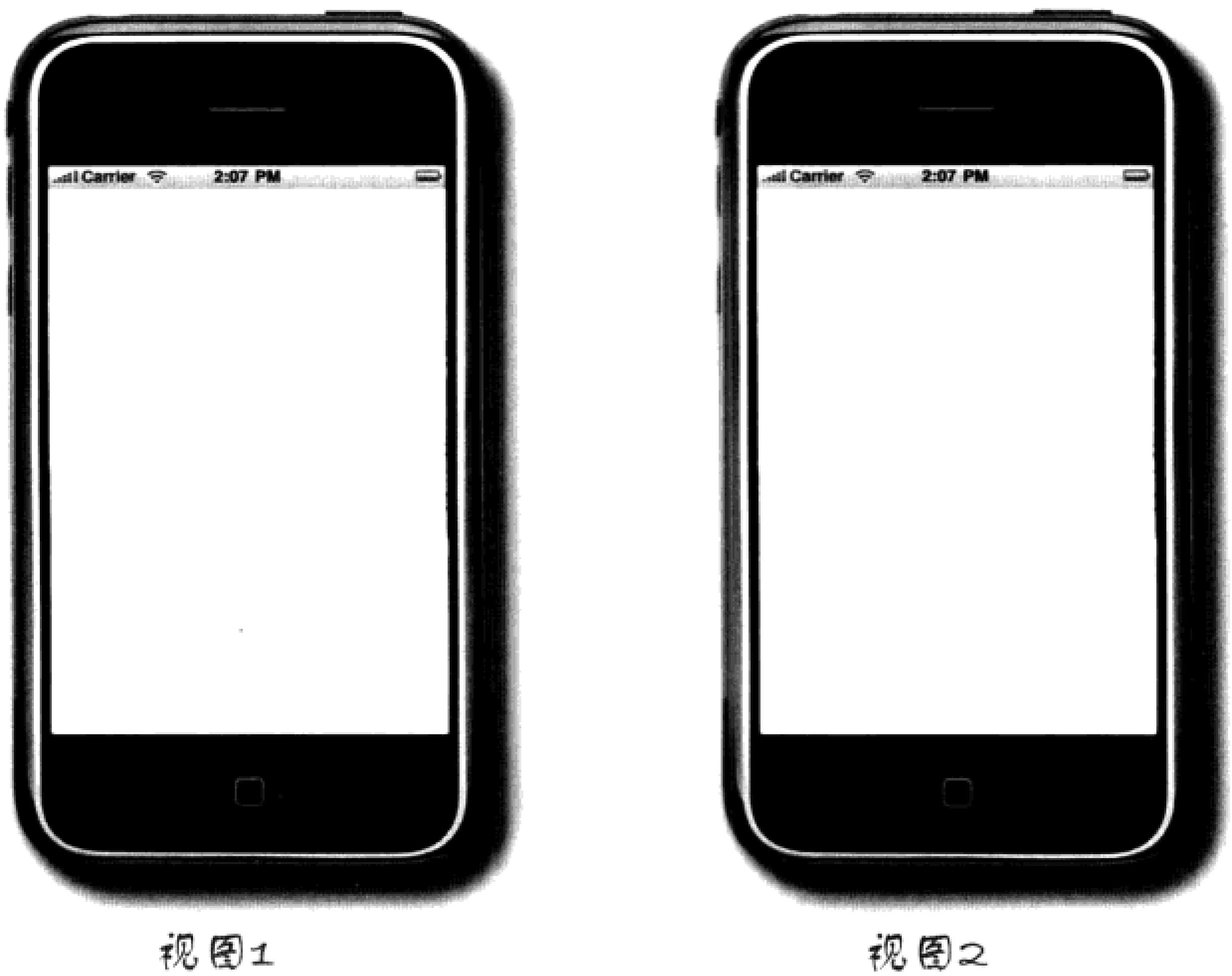
这一章是关于多视图应用的。那么对于一个酒吧助手应用，你需要哪些视图呢？

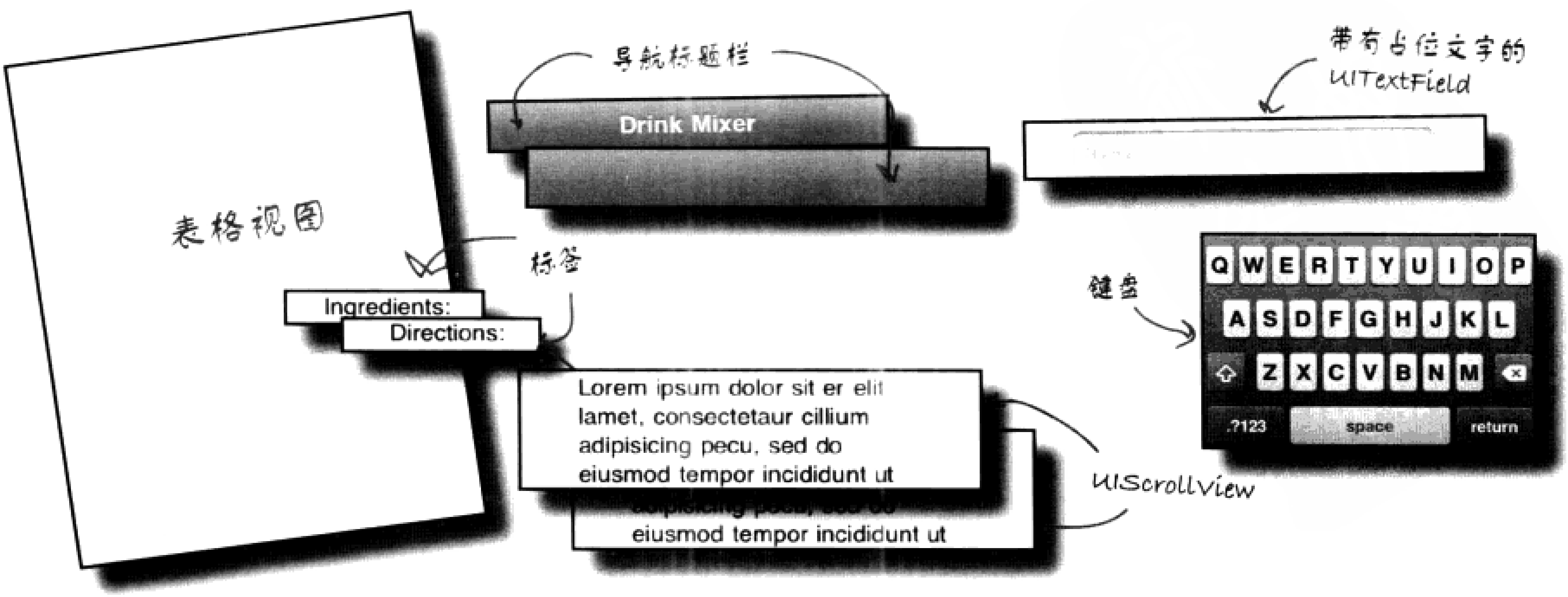
【拼吾爱-最新编程资源的分享下载站】



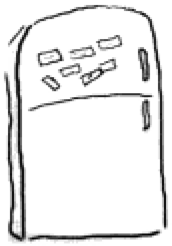
iPhone界面设计魔力贴

用下面显示的组件，为我们要在应用中使用的两个视图进行布局。



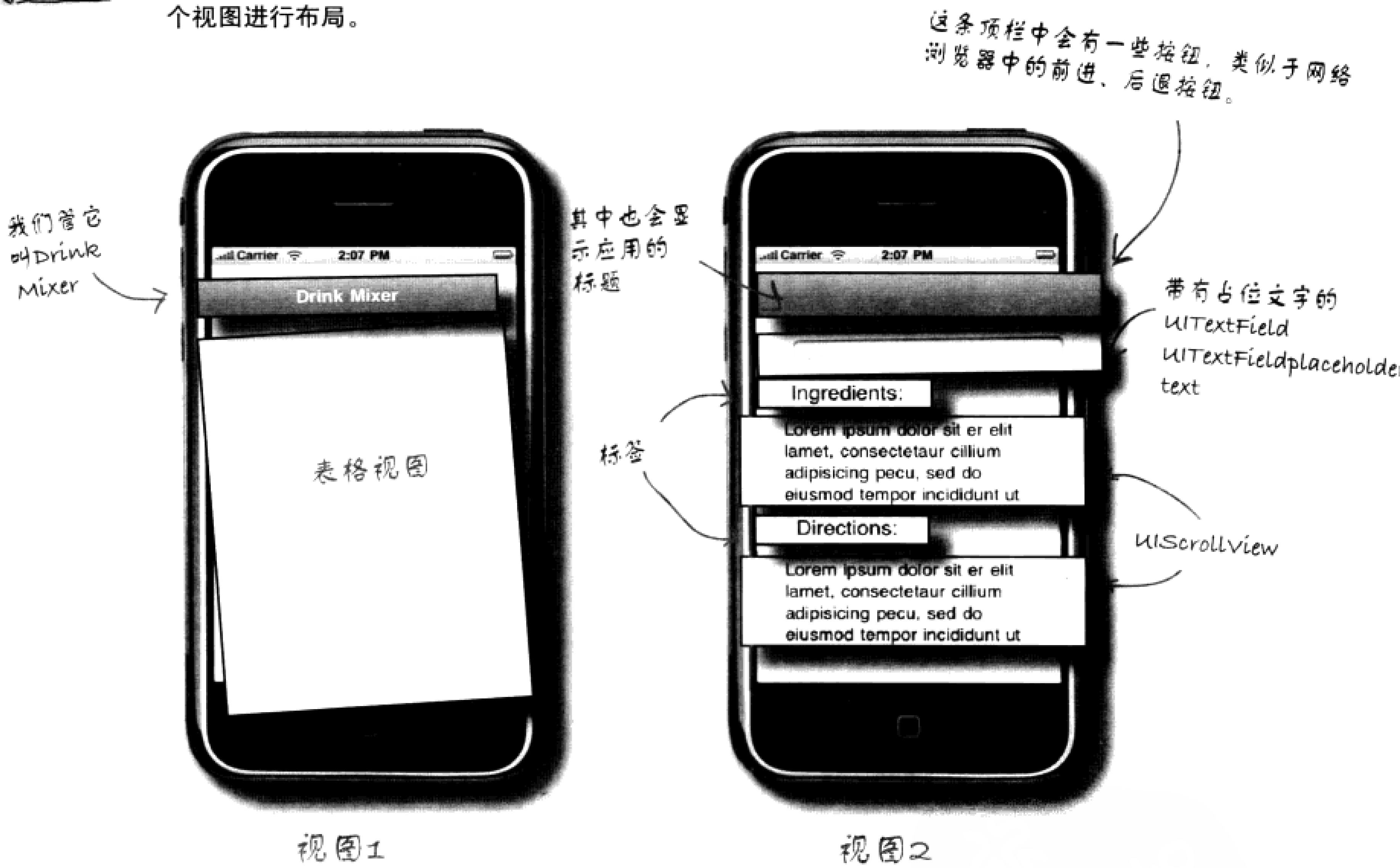


【拼吾爱-最新编程资源的分享下载站】



iPhone界面设计魔力贴答案

用下面显示的组件，为我们要在应用中使用的两个视图进行布局。



山姆需要一个饮料名称列表，并能查看饮料里面都有什么成份。他还需要知道各种成份的份量以及一些指导说明——哪些要加冰块，是否要摇晃或者搅拌，何时要点着一些东西，等等。所以对于我们的两个视图来说，首先会把饮料放在一个列表里（视图1），然后在山姆点击一种饮料的时候，就会显示细节（视图2）。

目前我们还不打算使用键盘——这是一个参考性质的应用，山姆只需要从里面读一些东西……



【拼吾爱-最新编程资源的分享下载站】

那么，怎么把视图结合在一起呢？

在你为我们的酒吧助手应用挑选模板之前，请先花一点时间斟酌一下，想要让用户怎样与饮料信息互动。我们打算用一个滚动列表显示饮料名称，当用户点击某一行时，我们就会用视图2，也就是细节视图，显示出饮料的详细信息。而当我们的用户看完了之后，他们会希望返回到饮料列表。



脑力举重

下面是应用可以使用的模板。你认为我们应该为 DrinkMixer 选择哪一种呢？

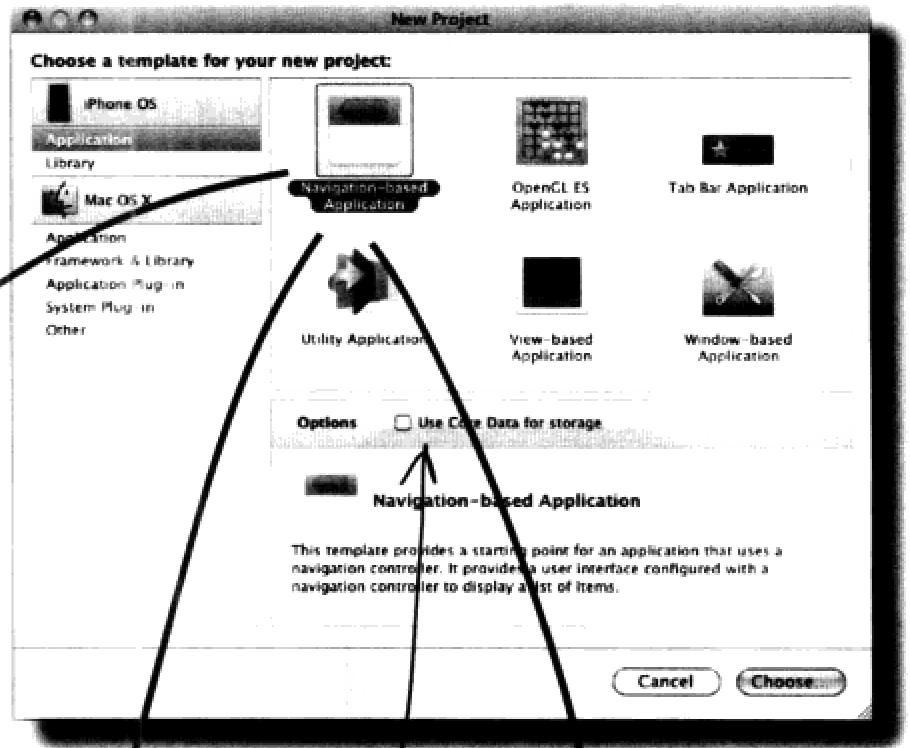
- ☐ 基于窗口的应用程序
- ☐ 基于视图的应用程序
- ☐ 工具应用程序
- ☐ 标签栏应用程序
- ☐ OpenGL ES应用程序
- ☐ 基于导航的应用程序

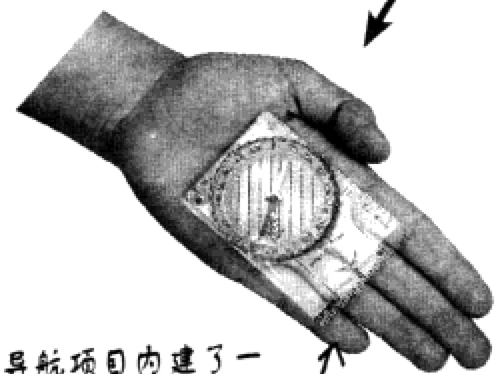
【拼吾爱-最新编程资源的分享下载站】

导航模板会把多视图整合到一起

对于这个应用，我们打算使用基于导航的项目。首先，请打开Xcode并在菜单中选择File→New Project。然后选择基于导航的应用程序并将其保存为DrinkMixer.proj。在选择模板时，请一定不要选中“Use Core Data for storage”这个选项。

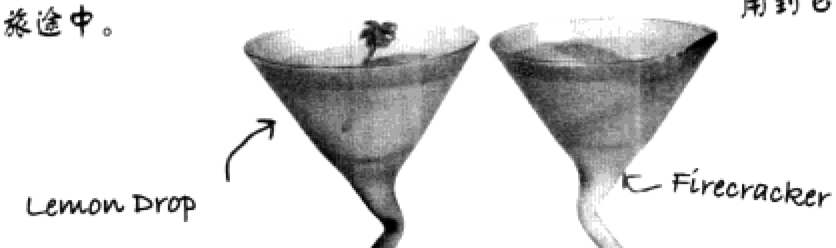
导航模板自带了许多内建的功能，包括：

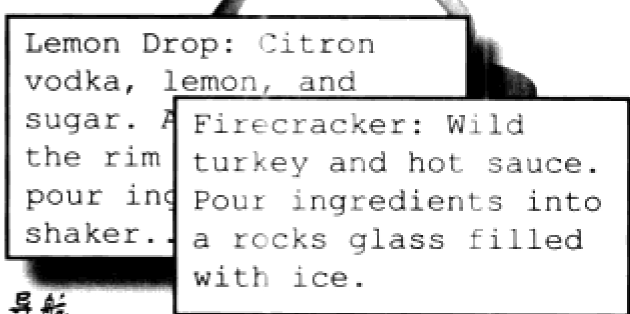


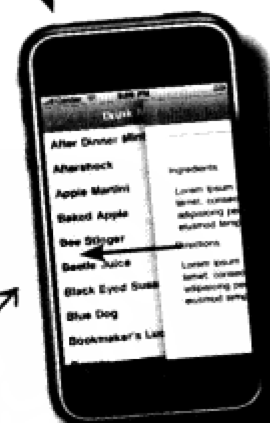


顾名思义，导航项目内建了一个导航控件。这个控件提供了后退按钮、标题栏，以及一份视图导航的历史，可以让用户不会迷失在浏览数据的旅途中。

不要选中Core Data选项。以后本书会用到它的。







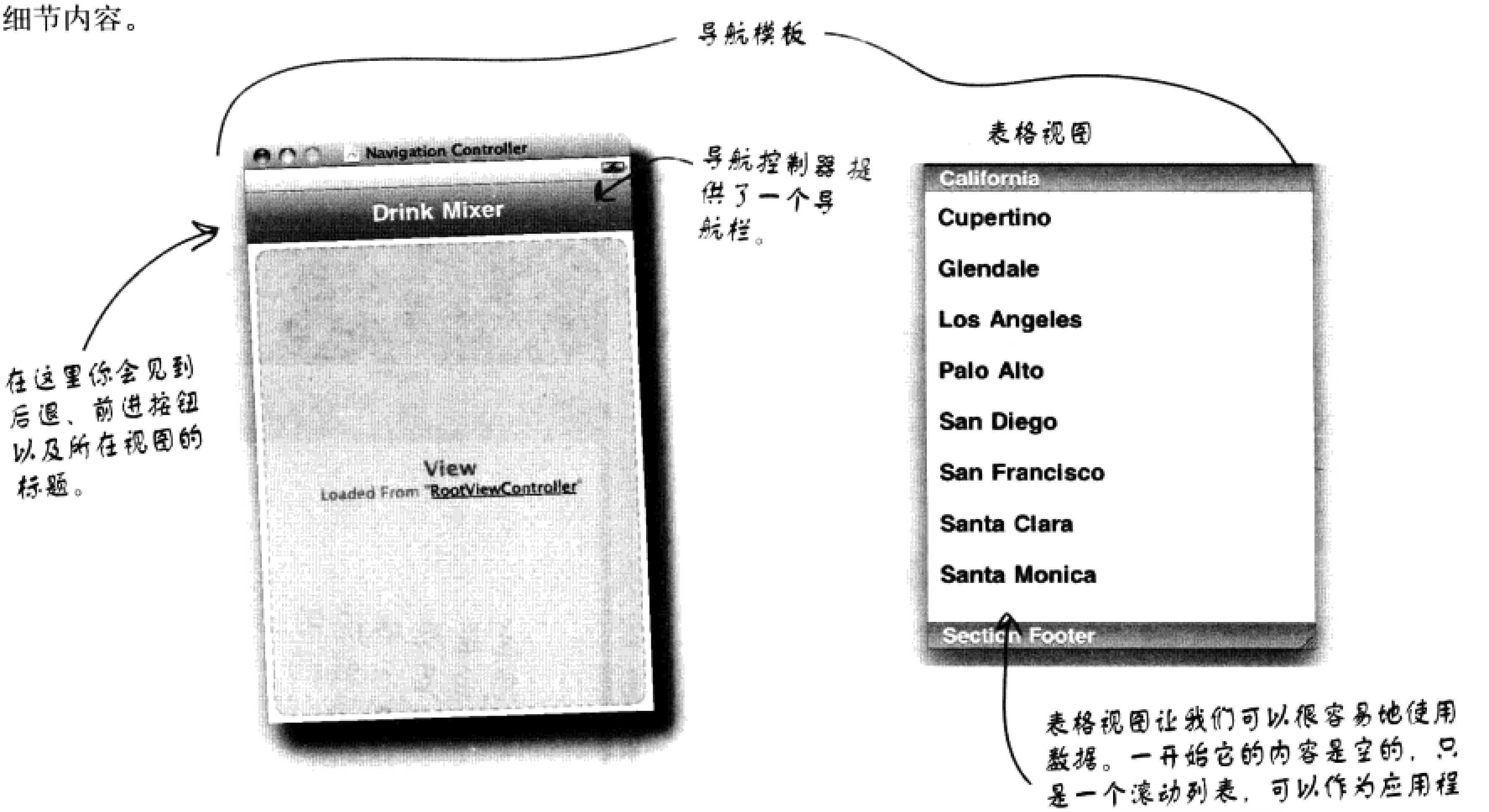
导航控制器会在视图之间提供动画形式的转场。

我们要组织一些分层数据。导航模板会帮助我们在数据间自如进退，让我们从一个表格视图开始吧。

【拼吾爱-最新编程资源的分享下载站】

从表格视图开始，学习使用导航模板

导航模板自带了一个导航控制器，以及一个根视图——也就是控制器在一开始会显示的视图。默认情况下，根视图会被设置为一个表格视图，这对于我们的应用来说非常合适。表格视图通常会用于对项目进行列表显示，每个项目被选中后都可以进一步显示更多的细节内容。



有问必答

问：如果导航模板是为了处理很多个视图准备的，为什么它仅仅自带了一个视图呢？

答： 多数基于导航的应用程序都是以一个表格视图作为起始点，然后以此出发显示更细节化的视图。有多少个细节视图、它们都是什么样，这些完全都是应用自己的事儿，所以你必须自己决定需要什么视图，自己加上这些视图。导航模板只会提供一个初始的表格视图，其他一概不管。

问：iPhone自带的应用程序中都有哪些使用了导航控制器呢？

答： 有联系人和邮件应用，这些都是核心的iPhone应用，它们都使用了这种设计方式。仔细研究一下电话上的这些应用，可以看到整个模板是如何实现的。如果要找一个比较干净的变种模式，则可以看一看短信（SMS）应用。这个应用也使用了一个导航控制器，但是却经常从“细节”视图开始，显示出最后一个发送或者接受短信的人的信息。

问：我必须用表格视图作为根视图吗？

答： 不必，这只是最常见的形式，因为这是一种很自然的方式，可以显示大量数据的概览，同时还可以让用户深入挖掘更多信息。表格视图的可定制度也非常高，所以某些应用可能看起来不像表格视图，比如说便签应用和iTunes Store。

【拼吾爱-最新编程资源的分享下载站】

运行测试

立即为主视图添加一个标题，并看一看空白表格视图是什么样子。在Interface Builder中打开 MainWindow.xib……



留神！

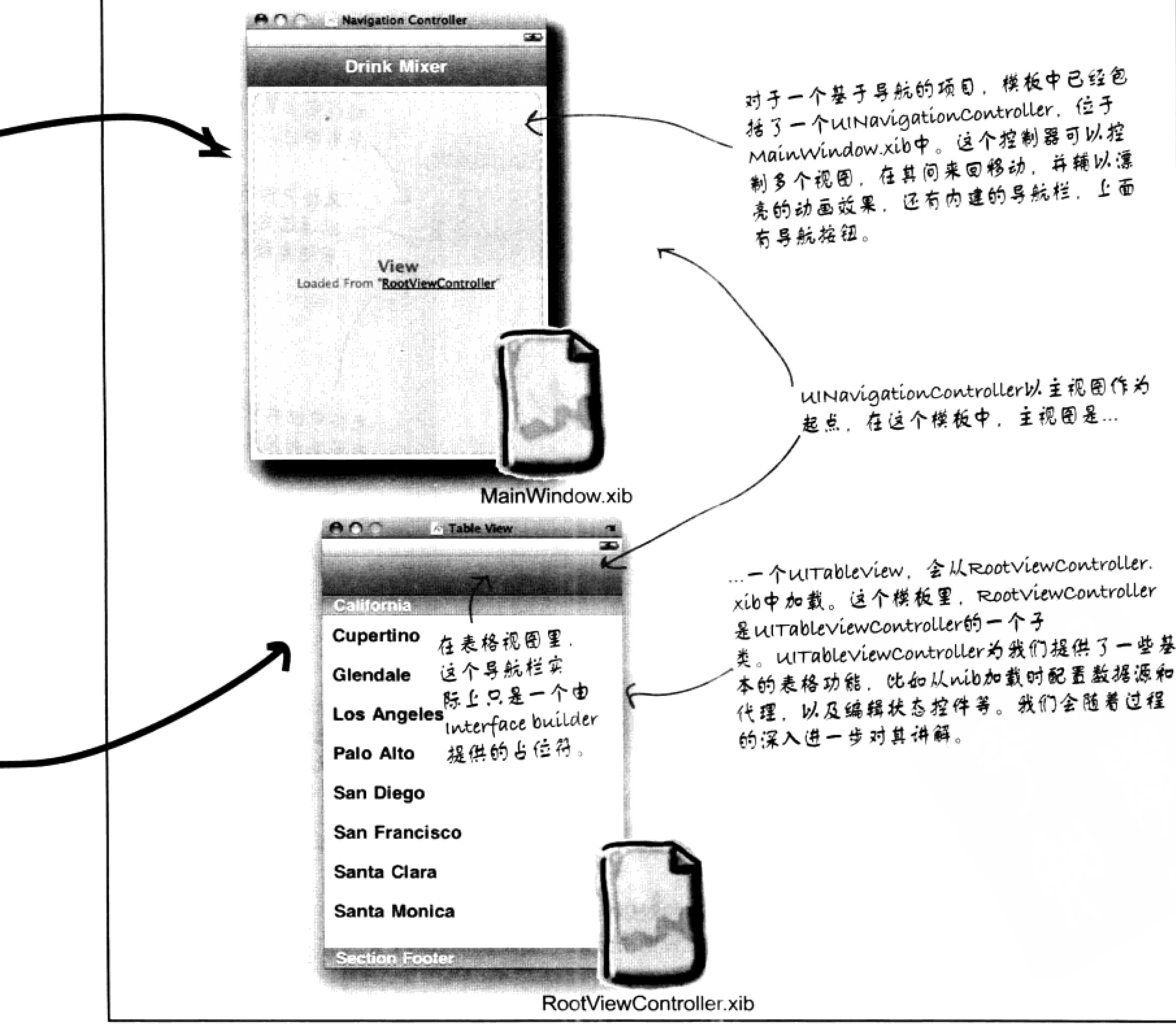
如果你在这里没有添加标题的话，一会儿就不会出现后退按钮了。

为应用的主视图设置了标题，也就意味着后来加入的视图上会自动出现一个后退按钮，可以回到主视图。

【拼吾爱-最新编程资源的分享下载站】

走近表格视图

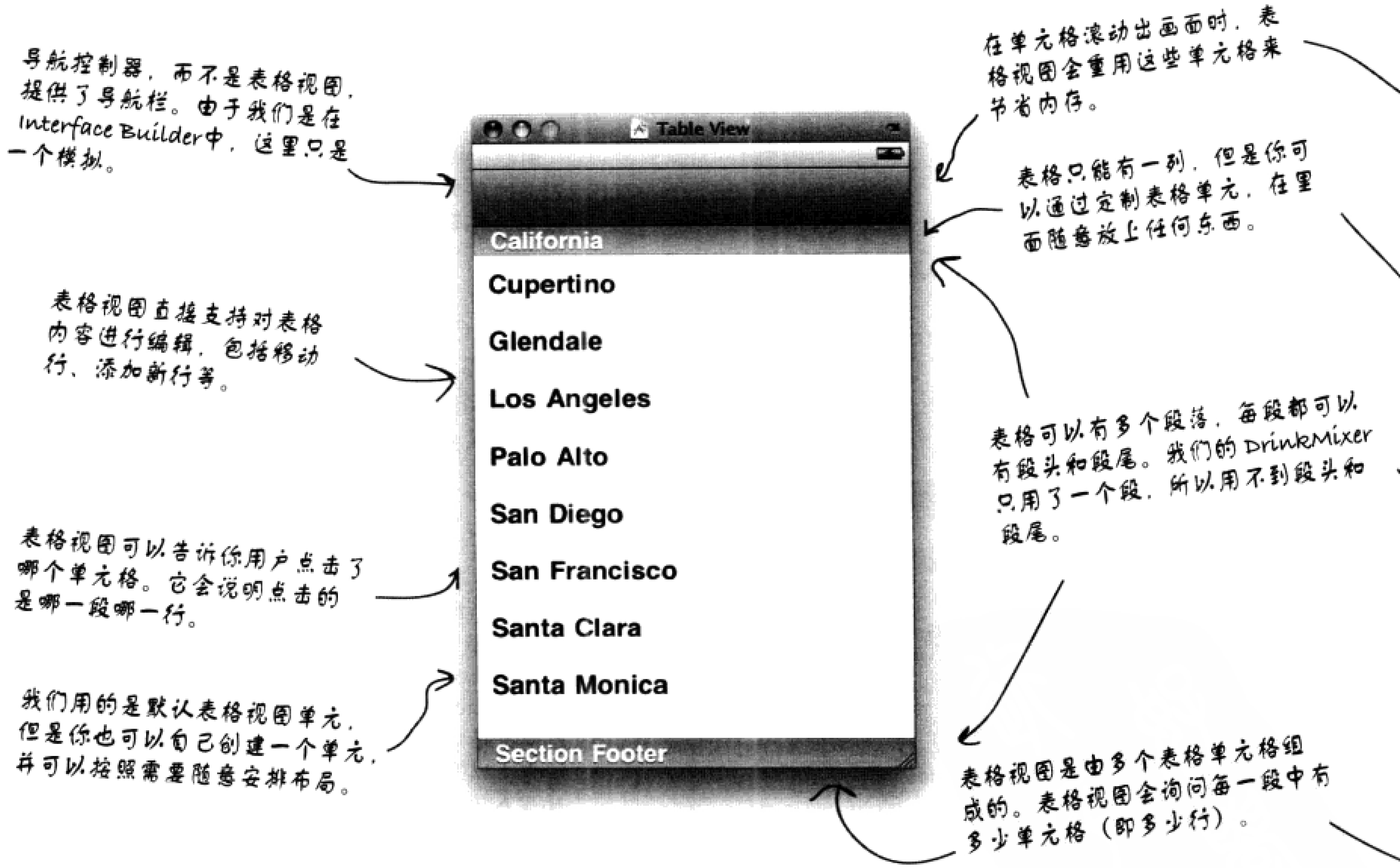
导航控制器和表格视图几乎总是被一起使用。当你选择了基于导航的项目作为模板之后，Xcode创建了一个不同于我们以往使用过的视图配置：



【拼吾爱-最新编程资源的分享下载站】

表格就是一组单元格

UITableView已经提供了许多我们马上就要用到的功能，但是它仍然需要了解我们想要显示的数据是什么，以及当用户与这些数据互动时该做些什么。这就是为什么要有数据源和代理。表格视图很容易就可以被定制，而且行为也被模板设置好了，必须通过与数据源和代理对话来得知要显示什么数据，显示多少行，使用什么样的表格单元等。



脑力练习

仔细观察你的设备上面的一些应用。你能找到的对表格视图进行定制的应用，哪些改动得最厉害？它们是否使用了段落？是否使用分组？ 它们是怎样对单元格进行布局的？

【拼吾爱-最新编程资源的分享下载站】

表格单元代码全接触

下面是从我们最新的RootViewController.m文件中摘录的一段代码。在这个文件里，我们会创建表格单元，并在其中填上饮料列表信息。

indexPath中包含有所需单元格的段落和行的编号。

当表格视图需要一个单元格时就会调用这个方法。

```
// Customize the appearance of table view cells.
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath
:(NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"Cell";
```

表格单元都有一个标识，这样当你要找一个单元来重用时，你能确认得到的单元格是正确的类型。

```
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellI
dentifier];
     if (cell == nil) {
         cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDe
fault reuseIdentifier:CellIdentifier] autorelease];
     }
```

在此我们查看表格视图中是否有可重用的单元格，这些单元格必须具有给定的标识。

如果没有任何可以重用的单元格，我们就创建一个新的。

```
        // Configure the cell.
cell.textLabel.text = [self.drinks objectAtIndex:indexPath.row];
     return cell;
}
```

这里我们用需要显示的特定饮料的信息来定制单元格中的文字。

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
     return 1;
}
// Customize the number of rows in the table view.
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInte
ger)section {
     return [self.drinks count];
}
```

这些方法告诉表格视图要有多少段落，每段中有多少行。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问： 这些单元格一开始是怎么进入到那个重用列表的呢？

答： 表格视图会处理这件事。当单元格滚动出画面时（从顶部或者底部），表格视图就会把不再需要的单元格放入队列。当它为特定行向数据源请求一个单元格时，你就可以查看这个队列，看是否有可以使用的单元格。

问： 我不明白表格标识是什么……必须是“Cell”吗？

答： 不必——那只是默认值而已。当你使用更复杂的表格视图时，可以根据要显示的数据，创建自定义的单元类型。你会在请求重用单元格的时候，用表格标识来确认其类型，表格视图会交给你所期望的类型。你可以随意定义标识——只要确保为每种唯一的单元格类型使用了唯一的名称就可以了。

拿起笔来

好了，是时候开始显示一些饮料了。你需要对RootViewController.h和RootViewController.m这两个文件做出一些修改。

1. **声明饮料数组。**
   在RootViewController.h中，声明一个名为drinks的数组，并辅以必要的属性声明，使用的语法与用于选取器的类似。

2. **实现并填充数组。**
   在RootViewController.m里，去掉viewDidLoad方法的注释，并扩展其内容，在其中创建这个数组，并加入这里的饮料列表。

3. **告诉表格需要多少行。**
   需要修改自动生成的代码才能告诉表格，行数要与数组中的饮料数目相同。请修改实现文件中位于下面这行之后的代码：`// Customize the number of rows in the table view.`

4. **填充表格单元。**
   请实现我们在前一页的表格单元全接触中所说的代码，这样就可以用数组中的内容来填充表格。



【拼吾爱-最新编程资源的分享下载站】

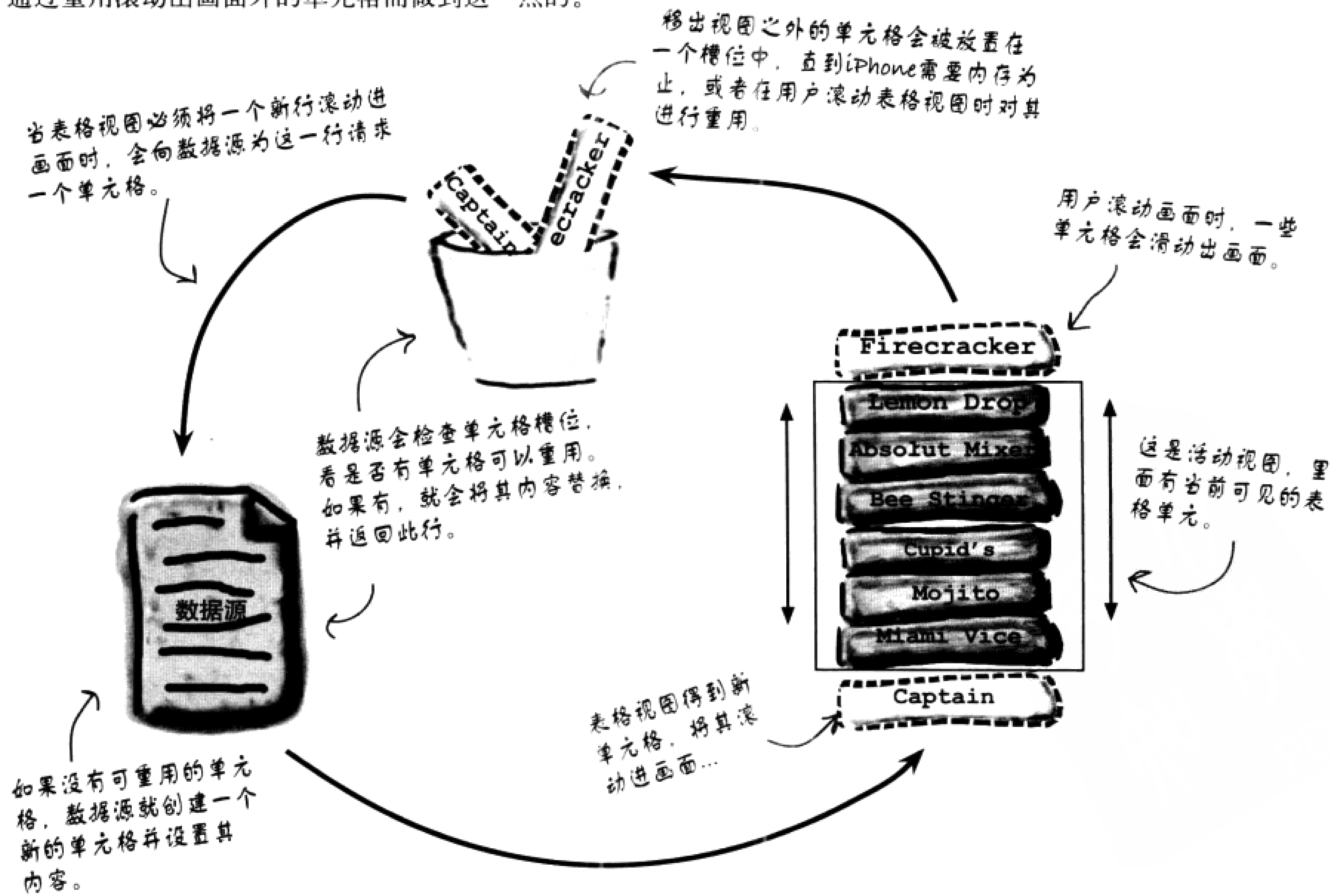


与iPhone上其他对象一样，UITableView必须考虑内存问题。

那么，它是如何在对消耗内存的顾忌与显示未知数量数据之间做出平衡的呢？答案是它把内容分解成了单元格。

每种饮料都有自己的单元格……诸如此类

UITableView必须显示的数据，只要能够填满iPhone的屏幕就行了——它并不会真正去在意潜在的全部数据量。UITableView是通过重用滚动出画面外的单元格而做到这一点的。



【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

好了，是时候开始显示一些饮料了。你需要对RootViewController.h和RootViewController.m这两个文件做出一些修改。

1 声明饮料数组。

```
@interface RootViewController : UITableViewController {
        NSMutableArray* drinks;
}
@property (nonatomic, retain) NSMutableArray* drinks;
@end
```

UITableViewController会替你处理数据源和代理的声明，所以在这儿就不需要声明了。

添加新数组drinks。

为drinks声明属性。

RootViewController.h

```
@synthesize drinks;
```

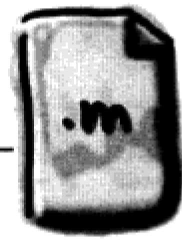
```
-
(void)dealloc {
        [drinks release];
        [super dealloc];
}
@end
```

RootViewController.m

【拼吾爱-最新编程资源的分享下载站】

2 **实现并填充数组。**

在RootViewController.m里，去掉viewDidLoad方法的注释，并扩展其内容。



RootViewController.m

```
- (void)viewDidLoad {
    [super viewDidLoad];

    NSMutableArray* tmpArray = [[NSMutableArray
alloc] initWithObjects:@"Firecracker", @"Lemon Drop",
@"Mojito",nil];
    self.drinks = tmpArray;
    [tmpArray release];

    // Uncomment the following line to display an Edit button in
the navigation bar for this view controller.
    // self.navigationItem.rightBarButtonItem = self.
editButtonItem;
}
```

这个方法一开始会被注释掉。

我们告诉你的初阶饮料。

3 **告诉表格需要多少行。**

```
//Customize the number of rows in the table view.
 - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInS
ection:(NSInteger)section {
    return [self.drinks count];
  }
```

这里以前返回的是0。

现在它会告诉表格视图，我们需要的行数与drinks数组中的成员个数一样。

4 **填充表格单元。**

```
// Configure the cell.
cell.textLabel.text = [self.drinks objectAtIndex:indexPath.row];
    return cell;
}
```

在这里我们用需要显示的特定饮料的信息，指定了单元格中的文字。

【拼吾爱-最新编程资源的分享下载站】

运行测试

现在已经一切就绪。请保存修改，构建并运行应用，你就会见到3种饮料出现在应用的主视图中。



试一下——列表也会滚动的！

这些看起来非常不错。我会马上把我们的完整列表用电邮传给你——一共40种饮料……



【拼吾爱-最新编程资源的分享下载站】

有问必答

问：你提到了表格视图的数据源和代理，但是为什么我不用像使用UIPickerView那样声明一些东西呢？

答： 问得好。通常你都得进行声明，但是我们使用的基于导航的模板已经把这些都安排好了。要想知道是到底怎么回事，可以看看RootViewController.h文件。你会发现这是一个UITableViewController的子类，而表格视图控制器同时遵循了UITableViewDataSourceProtocol和UITableViewDelegateProtocol这两个协议。如果你查看RootViewController.xib文件，会发现表格视图的数据源和代理都已经被设置为RootViewController。如果我们没有用模板，你就必须自己完成这些设置（我们会在第7章中重温这一内容）。

问：我发现文中用到了一个NSMutableArray。这是因为我们必须对它进行初始化吗？

答： 不是的——不管是NSArray还是NSMutableArray，都可以在创建时进行初始化赋值。我们之所以使用NSMutableArray，是由于打算以后对这个数组的内容进行操作。马上我们就会讲到这一点。

问：在创建饮料数组的时候，饮料名称最后面的nil是做什么用的？

答： NSMutableArray的初始化方法可以接受可变数目的参数。所以它使用nil来表示到达了参数的末尾。 数组中的最后一个元素会是nil前面的值——nil并不会被添加到数组中。

问：能不能再跟我说一次饮料名称前面的@符号含义？

答： @符号是一种简便的创建NSString的方法。NSArray中数组的存储内容是对象，所以我们需要把名字从文本(char*)转换成NSString。把一个@放在文字常量前面，就可以完成这一转换。

问：当我们定制表格视图的单元格时，使用了属性cell.textLabel。还有其他的标签吗？cell.textLabel和cell.text之间有什么区别呢？

答： 在iPhone 3.0之前，默认类型的单元格中只有一个标签和一组披露指示器，完全由单元格自己处理。你只能把需要的文字设置到cell.text属性上。几乎每个人都希望能在表格单元中多显示一点信息，所以在 iPhone 3.0中，苹果公司添加了几种不同的风格与标签布局。如此一来，他们也随之为不同的文字区域引入了专门的属性，比如textLabel、detailLabel等，并且将原来的cell.text属性列为准备废弃。你不应该继续在应用中使用cell.text了——苹果公司很可能会在将来某个时候将其删除。稍后在本章中我们会进一步介绍其他标签。

问：你提到了我们可以用段头和段尾——怎么去指定这些呢？

答： 也是由数据源负责提供这些信息。你可以实现数据源中的一些可选方法，根据段落编号来返回段头和段尾的标题。这些方法的工作方式与我们的cellForRowAtIndexPath方法类似，唯一区别是返回值为字符串。

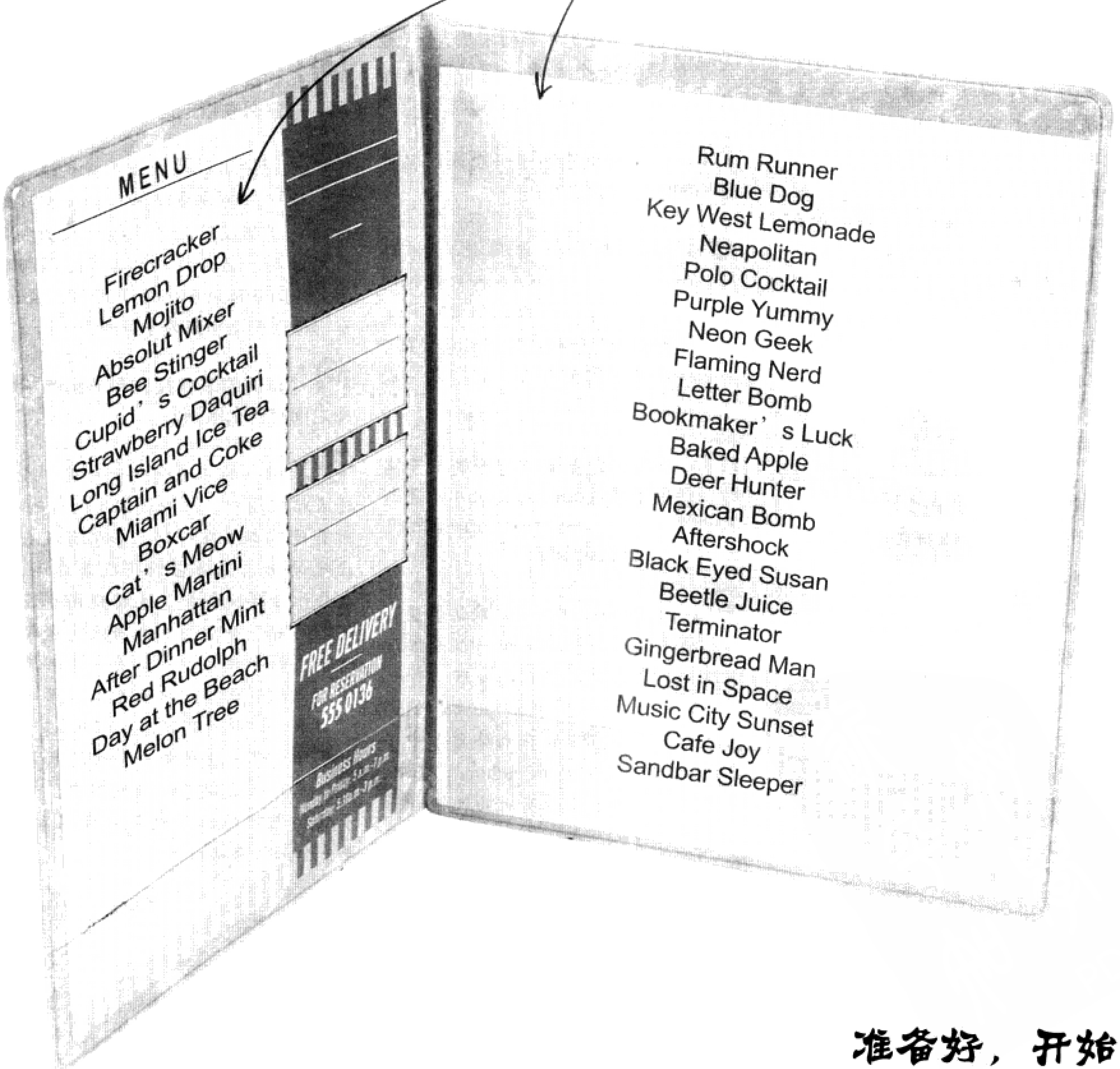
问：普通的表格视图和分组表格视图之间有什么区别？

答： 唯一的不同就在于外观。在普通表格视图中，比如我们用的这一个，所有段落都挨在一起，如果有段头和段尾的话，会由它们将段落分隔开。在分组表格视图中，表格视图会在段落之间留一些空间，并将段头和段尾以较大的字母显示。请看一看联系人列表，然后选择一个联系人。第一个视图中，你的所有联系人都列在一起，由字母分开，这就是一个普通表格视图。而后来出现的细节视图中，电话号码和电邮地址等就是分开显示的，是一个分组表格视图。

【拼吾爱-最新编程资源的分享下载站】

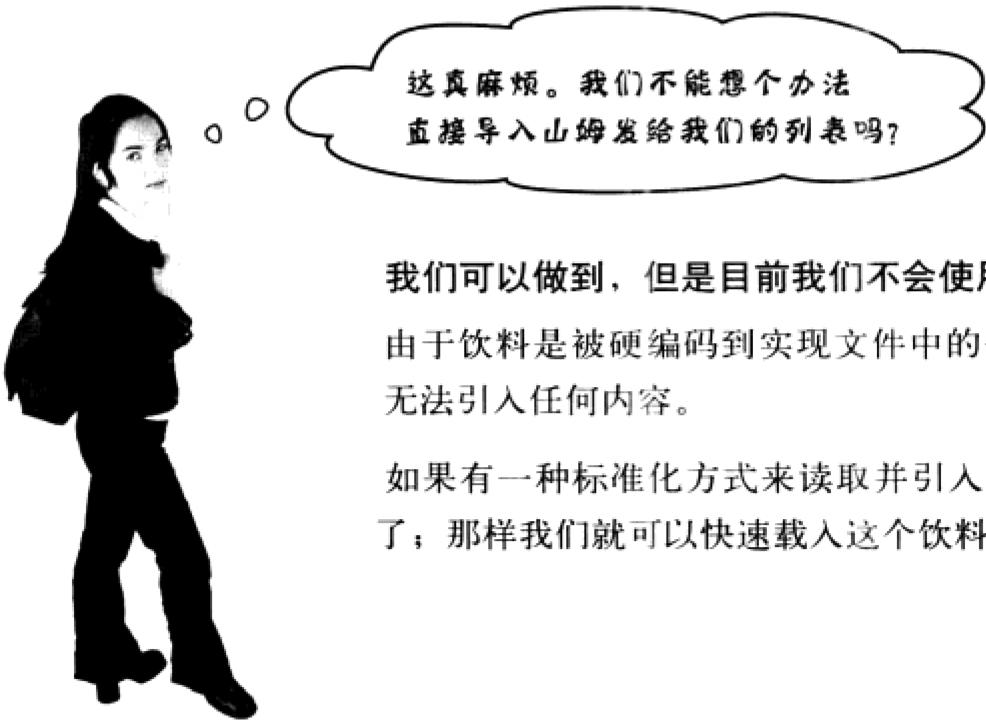
再来几种饮料

Head First休闲吧的饮料单中有40种鸡尾酒。



准备好，开始打字吧……

【拼吾爱-最新编程资源的分享下载站】



我们可以做到，但是目前我们不会使用这种方法。

由于饮料是被硬编码到实现文件中的一个数组里的，我们无法引入任何内容。

如果有一种标准化方式来读取并引入数据的话，那就太好了；那样我们就可以快速载入这个饮料列表了。



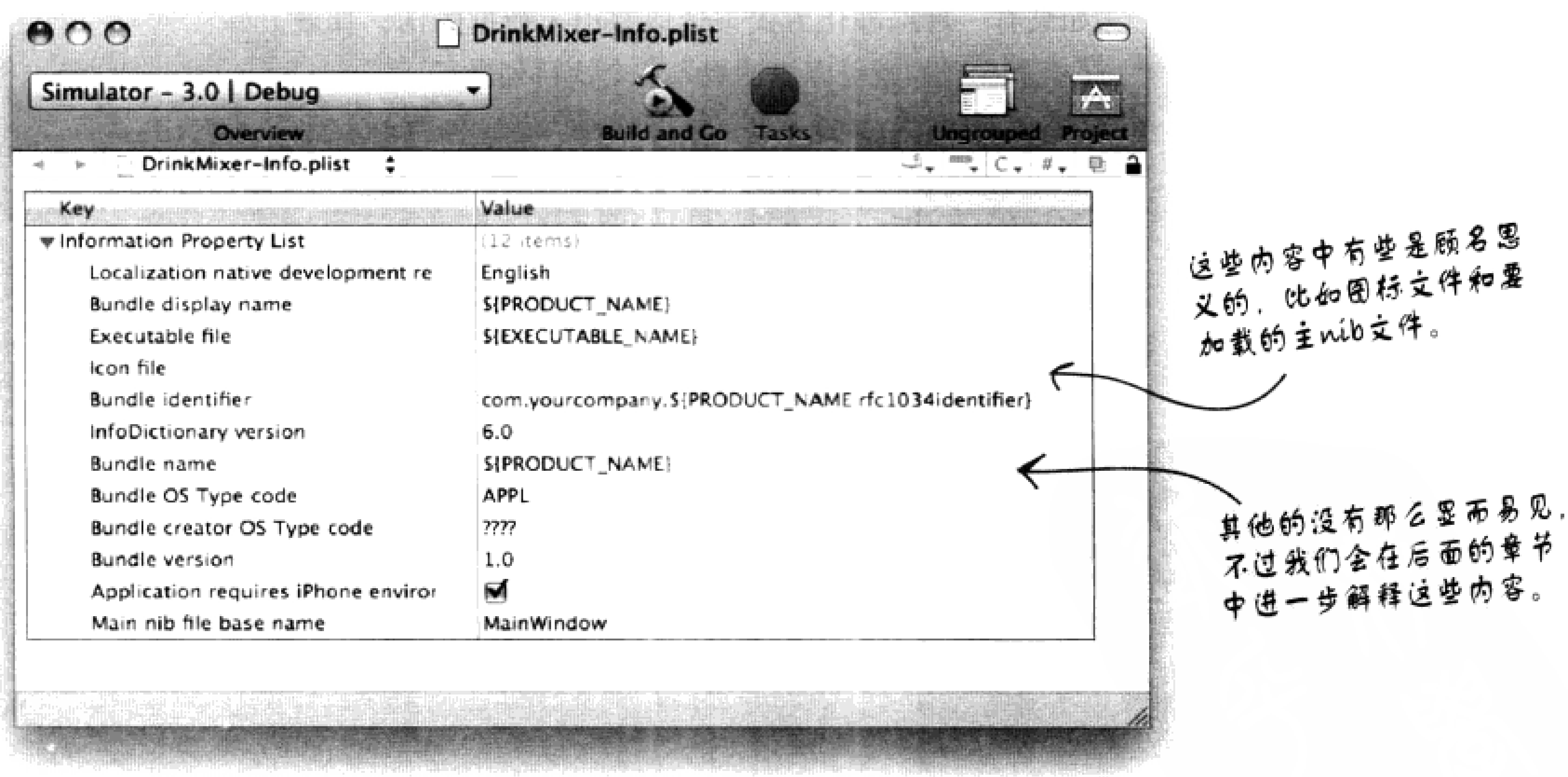
脑力举重

我们可以做些什么呢？我们需要一种方法加速整个过程。

【拼吾爱-最新编程资源的分享下载站】

Plist是一种保存和载入数据的简便方法

Plist代表“属性列表”，而且在OS X中已经存在很久了。实际上，在你的应用程序中已经用到很多plist了。我们已经用过了最重要的plist——DrinkMixer-Info.plist。这是在你启动项目的时候由Xcode创建的，里面除了指定应用的图标之外，还存储了其他一些东西，比如应用程序启动时加载的主nib文件、应用程序版本等。像其他文件一样，Xcode可以创建并编辑这些plist文件。点击DrinkMixer-Info.plist，就可以看到里面有什么内容。



【拼吾爱-最新编程资源的分享下载站】

内建类型可以自动在plist中保存和加载

我们曾经用过的所有内建类型，比如NSArray和NSString，都可以自动从plist加载和保存。我们可以利用这一点，把饮料列表从源代码中移走。

我们会把这里的饮料列表从源代码中移走，转而放到一个plist中去…

```
- (void)viewDidLoad {
    [super viewDidLoad];

        NSMutableArray* tmpArray = [[NSMutableArray
alloc] initWithObjects:@"Firecracker", @"Lemon Drop",
@"Mojito",nil];
        self.drinks = tmpArray;
        [tmpArray release];

    // Uncomment the following line to display an Edit
button in the navigation bar for this view controller.
    // self.navigationItem.rightBarButtonItem = self.
editButtonItem;
}
```



练习

在引入山姆的列表之前，让我们创建一个具有相同格式的示例plist。我们会首先确认这个列表工作正常，然后再把山姆的列表放进来。

1 **创建空白plist。**

回到Xcode中，打开Resources文件夹。在Resources上单击右键并依次选择Add → New file、Mac OS X Resource和Property List。给新的列表取名叫做DrinkArray.plist。

一定要选择Mac OS X下的“Resource”——plist没有列在iPhone资源里。

2 **格式化 plist 并填充内容。**

打开这个文件，并把根类型改为数组，条目类型改为字符串。然后你就可以把饮料列表里的名字填进去了。



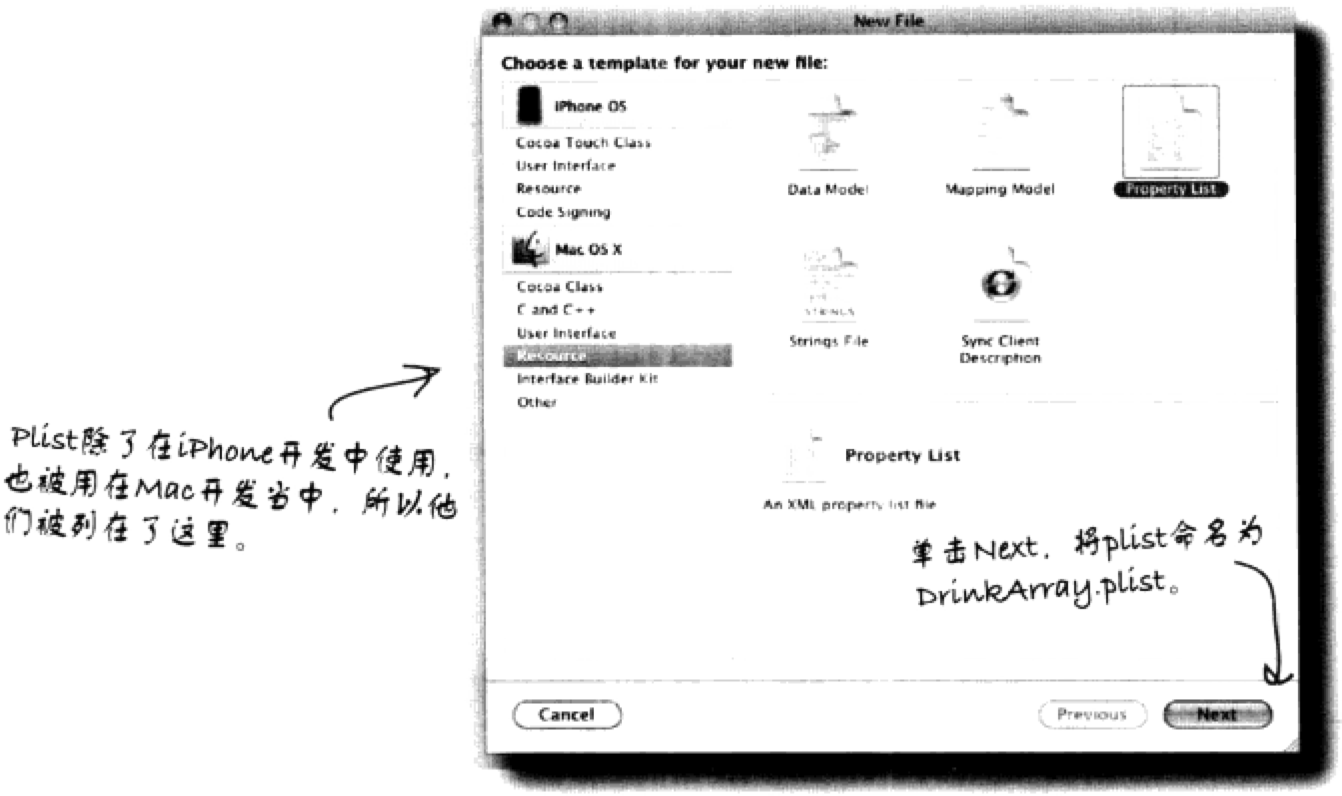
【拼吾爱-最新编程资源的分享下载站】

练习答案

用这个自己创建的示例列表，我们就可以在开始使用大型列表前先进行测试。

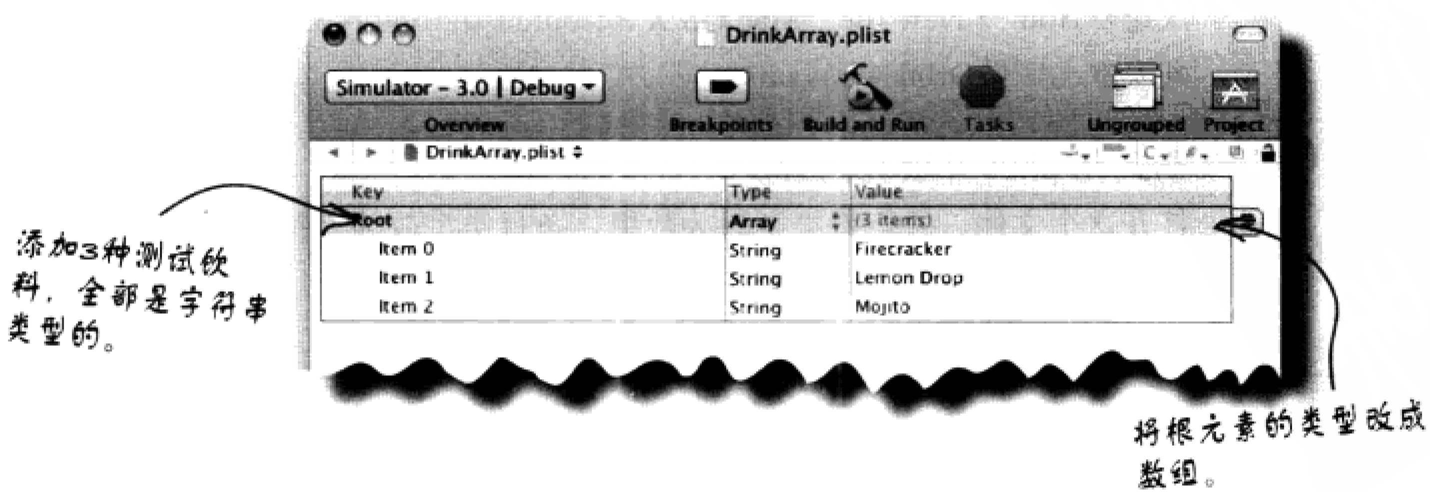
1 创建空白plist。

回到Xcode中，打开**Resources**文件夹。在**Resources**上单击右键并依次选择**Add → New file**、**Mac OS X Resource**和**Property List**。给新的列表取名叫做**`DrinkArray.plist`**。



2 格式化 plist 并填充内容。

打开这个文件，并把根类型改为数组，条目类型改为字符串。然后你就可以把饮料列表里的名字填进去了。



【拼吾爱-最新编程资源的分享下载站】

数组（及其他一些类型）内建了对plist的支持

修改数组的初始化代码来让它使用plist是非常容易的。大多数Cocoa中的群集(collection)，比如NSArray和NSDictionary都内建有在plist中序列化/反序列化的支持。只要你使用的是内建类型（比如其他群集、NSString等），就可以直接让数组根据一个plist进行初始化。

万事俱备，只欠告知数组使用哪一个plist了。这一点我们可以通过项目的资源束来完成，资源束是用来处理应用程序特有的信息与文件的。

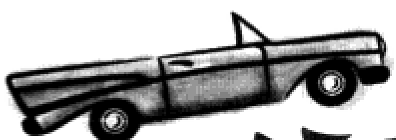
```
-(void)viewDidLoad {
  [super viewDidLoad];
    NSString *path = [[NSBundle mainBundle] pathForResource:@"DrinkArray" 
ofType:@"plist"];
    NSMutableArray *tmpArray = [[NSMutableArray alloc] 
initWithContentsOfFile:path];
    self.drinks = tmpArray;
    [tmpArray release];
    ...
```

向应用束请求名为DrinkArray的plist的路径。

用plist的内容初始化数组。



RootViewController.m



运行测试

做完这两件事之后，请继续构建并运行应用。这时候它看起来应该是一样的，也只有3种饮料。



即食plist

一旦这个列表工作正常了，请光临网址http://www.headfirstlabs.com/iphonedev并下载DrinkArray.plist文件。这是Head First休闲吧的完整饮料列表。用这个文件覆盖你的测试列表，重新构建DrinkMixer，然后再试一次！

【拼吾爱-最新编程资源的分享下载站】

运行测试

现在整个列表都出现了！

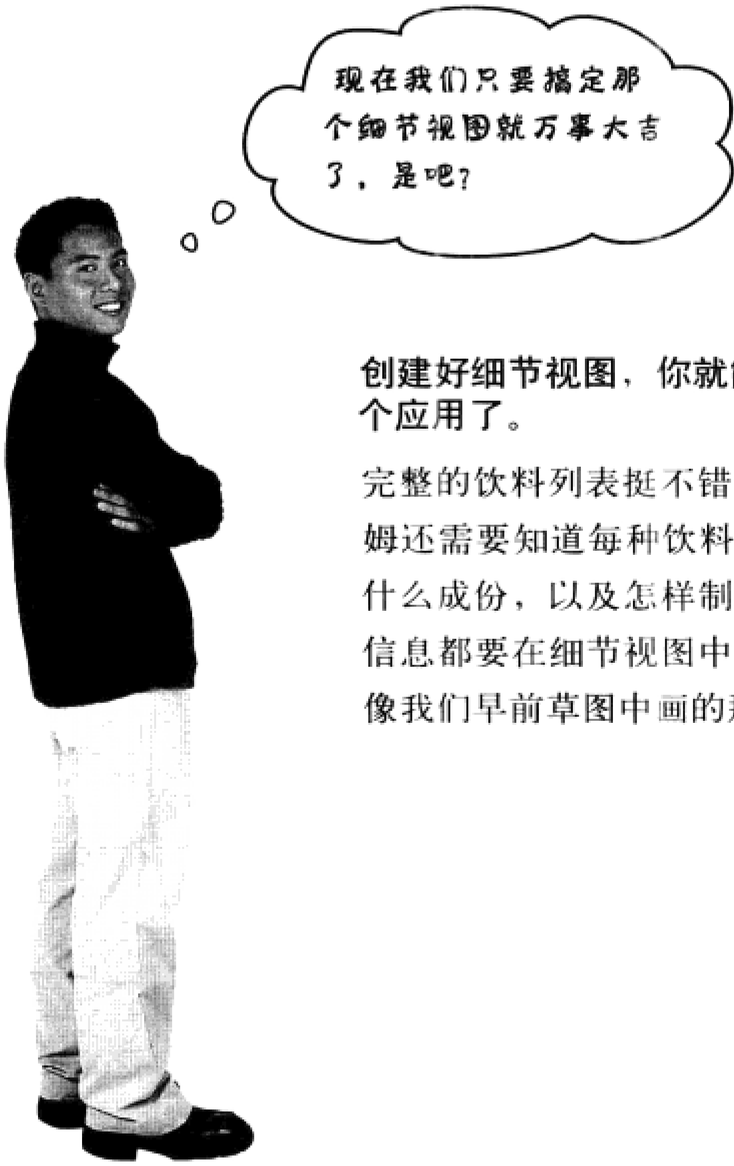


把饮料名从代码中挪到一个外部文件中，让你可以不用碰代码就能修改这个饮料列表。

在iPhone上，plist只是存储数据的方法之一——稍后本书中还会讲解其他方式。

Plist对于内建类型非常好用。如果你打算使用自定义类型，很可能就得考虑使用其他方式了。

【拼吾爱-最新编程资源的分享下载站】



创建好细节视图，你就能完成整个应用了。

完整的饮料列表挺不错，但是山姆还需要知道每种饮料里面都有什么成份，以及怎样制作。这些信息都要在细节视图中显示，就像我们早前草图中画的那样。

脑力举重

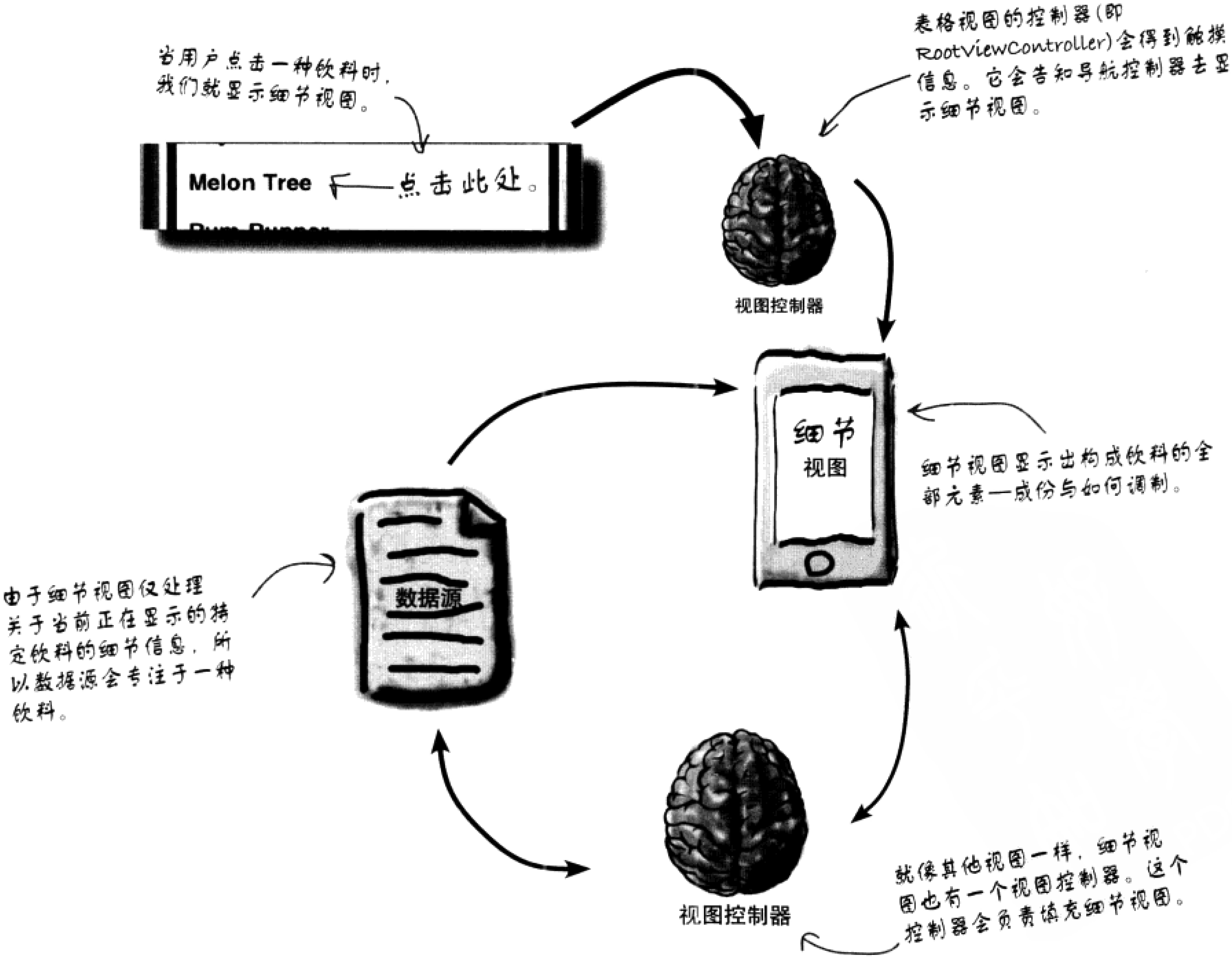
我们该怎样从列表视图转换到细节视图呢？而又打算如何显示这些细节呢？

【拼吾爱-最新编程资源的分享下载站】

用细节视图深入发掘数据

稍早些时候，我们将DrinkMixer定位为生产力应用，并选择了导航控制器来处理我们的分层数据。我们加载了一份长长的列表，但是山姆现在还需要每种饮料的详细信息：成份是什么，如何调制，等等。现在我们将要使用这个导航控制器来为列表中的饮料显示一个更详细的视图。

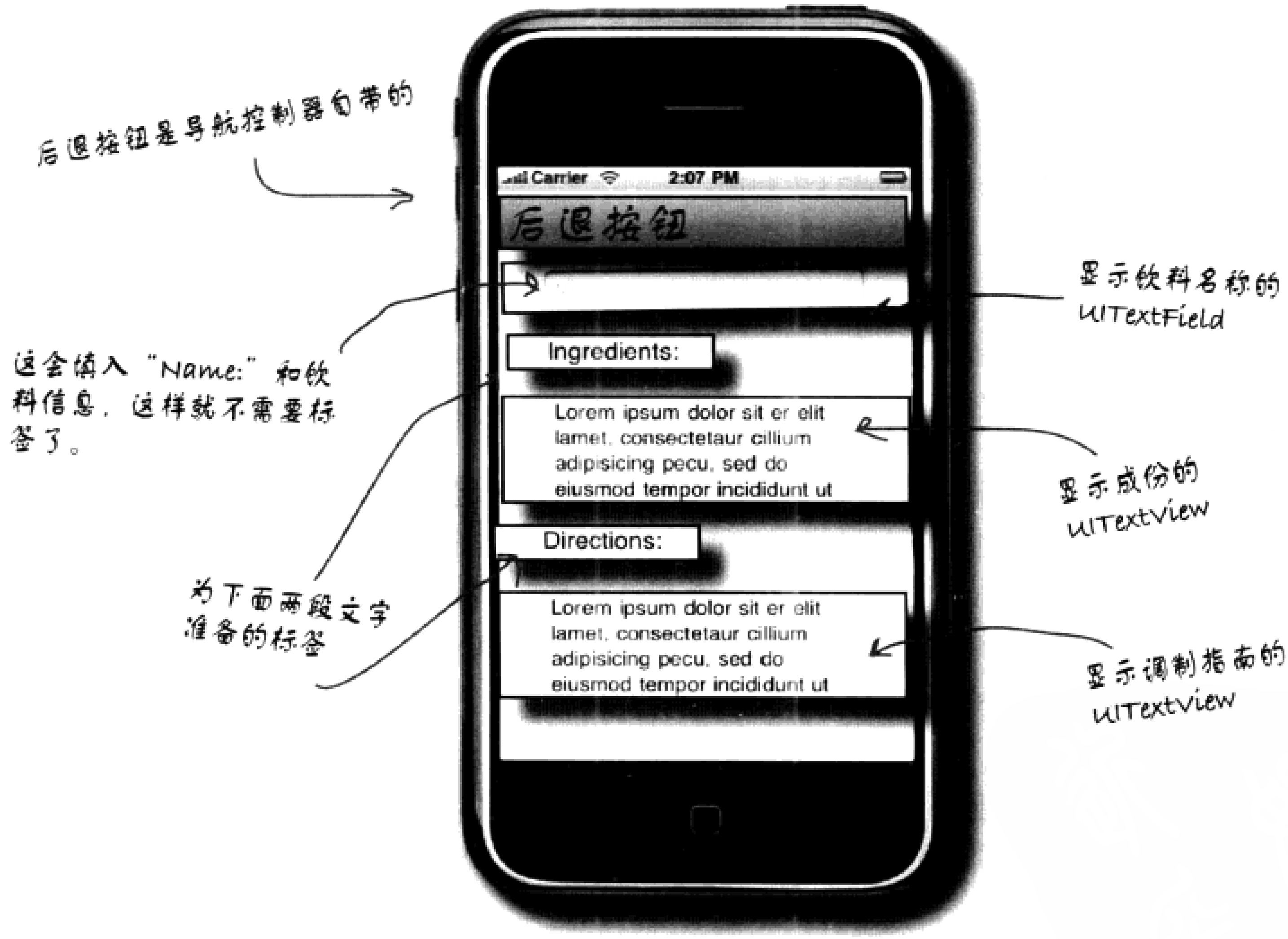
表格视图的标准使用模式是：当用户点击一个表格单元时，就显示关于该条目的更多信息。我们会用这种模式让用户选择一种饮料，然后显示我们的细节视图。细节视图与其他视图遵循同样的模式：



【拼吾爱-最新编程资源的分享下载站】

仔细考察细节视图

早前我们曾勾勒出了细节视图的草图——但是我们需要更仔细地考虑到底要打造出什么样的视图。

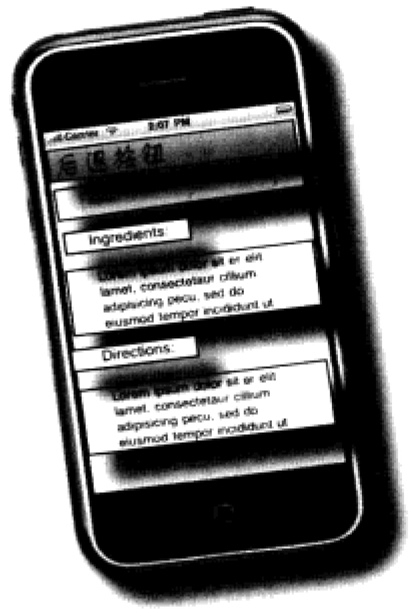


现在开始构建应用……

【拼吾爱-最新编程资源的分享下载站】

长篇练习

现在你已经知道怎么做了。请开始打造你的细节视图，创建所需的文件和代码，然后将它们在Interface Builder中联系、整合到一起。开动！



1 创建所需文件。

要创建新的视图，需要一个新的*.xib文件，以及进行相关支持的头文件和实现文件。文件类型是Cocoa Touch Class，并且是UIViewController的子类。

2 在Interface Builder中为新视图进行布局。

使用控件库，将需要的元素拖曳到视图中，按照我们先前勾画的模样打造这个视图。

这里给一点提示：要为导航控制器留出空间。打开Inspector，在Simulated Interface Elements段落中，从Top Bar下拉框中选择Navigation Bar。这样就可以确保将你的视图布置在导航栏的下方。

3 编写代码，完成新字段的声明和输出口定义。

你需要同时更改DetailViewController.h和DetailViewController.m这两个文件。请将新的文本字段命名为nameTextField、ingredientsTextView以及directionsTextView。

4 将细节视图与新的输出口连接起来。

就像我们对InstaTwit做的一样，用Interface Builder让新视图工作起来。

5 将文本字段设为不可更改。

用查看器找到点选框，将字段设置为不可更改。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：我们一直把数据源、视图和视图控制器画成独立的事物，但是我们又把它们合在一起放在同一个类里面。这到底是在搞什么名堂？

答： 这都是模式惹的祸。一般来说，在nib里会定义一个视图，还有一个用来支撑视图的控制器，以及一组工作时使用的数据。这些东西是不是要结合进一个类中，视应用程序的复杂程度而定。如果你没有使用Interface Builder，完全可以不管这些深奥的东西，直接在一个类中用编程的方式创建视图。我们稍后会在本书中进一步展示这种方法。然而从理论上来说，仍然有一个视图存在，它会在发生某些情况时调用视图控制器。同样地，你通常也会在某个地方实现一个或多个数据源协议，向你的视图提供数据。

问：为什么我们必须把*.xib文件移到Resources组里去呢？

答： 你并不必如此，但是我们建议这样做，好让代码整洁有序。不同的开发者使用不同的分组方式，可能会有“User Interface”、“Business Objects”、“Data Objects”等。Xcode实际上并不在乎这些；这些分组的重要性仅仅体现在让你了解代码是如何组织的，让你可以找到想找的东西。重用一种其他人也认识的结构是很好的行为，这样别人可以快速地看懂你的代码，而你也可以理解他们的。在本书中，我们使用模板的默认分组。

问：还有其他什么方法可以保存数据吗？

答： 有的，还不少呢。我们会在这本书里用不同的项目来介绍一些比较常见的方法。你现在用的这种plist是最简单的，但是对保存和加载的对象有限制。这不是说这种方法不好；如果它能满足你的需要，就是一个良好的解决方案——只不过它限制太大，无法适用于所有情况。还有一种序列化的方法，叫做NSCoding，可以很好地用于定制对象，但是要在不同版本间移植就成了问题。iPhone还支持用SQLite在数据库中保存和加载数据。在过去，如果你有大量的数据，或者要对数据进行搜索访问而又不想把数据全放进内存里，这是最好的方法。然而，随着iPhone 3.0，苹果公司引入了Core Data。Core Data是一个非常强大的框架，为持久化提供了一个面向对象的包装层，而且使用SQLite能得到的所有益处它几乎都具有。这个框架想入门绝对要花费一些精力，但确实很强大。以后我们会用它来打造一个应用。

问：为什么在名称字段上不用标签？

答： UITextField允许在字段为空时，在其中显示一小段占位文字。我们不想再加一个名字的标签，那就把屏幕空间都用光了，所以选择了使用占位文字。如果屏幕上显示的文字含义很明显，就可以考虑用占位文字。

问：那为什么对于成份和调制指南我们就不这么做了呢？

答： 也可以这么做，但是这些字段里面都包含有多行文字，我们想用标签把它们分开，明确地说明它们的意思。说到底，这个决定是基于美观和可用性方面的考虑，而不是技术方面的问题。

关键要点

- 生产力应用非常适合用于分层数据。
- 导航控制器是一种非常好的管理多视图的方法。
- 表格视图通常与导航控制器配合使用。
- iPhone的表格只有一列，但是可以显示自定义的单元格。
- 表格需要一个数据源和一个代理。
- 多视图通常会使用多个*.xib文件。

【拼吾爱-最新编程资源的分享下载站】

练习答案

这里是创建新的细节视图需要的所有信息。做完这个练习之后，你就应该得到一个可以工作的（但还是空白的）细节视图了。

1 **创建所需文件。**

我们需要一个新的.xib来存放细节视图。要想从头创建的话，可以回到Xcode，并从菜单中依次选取**File→New File**……

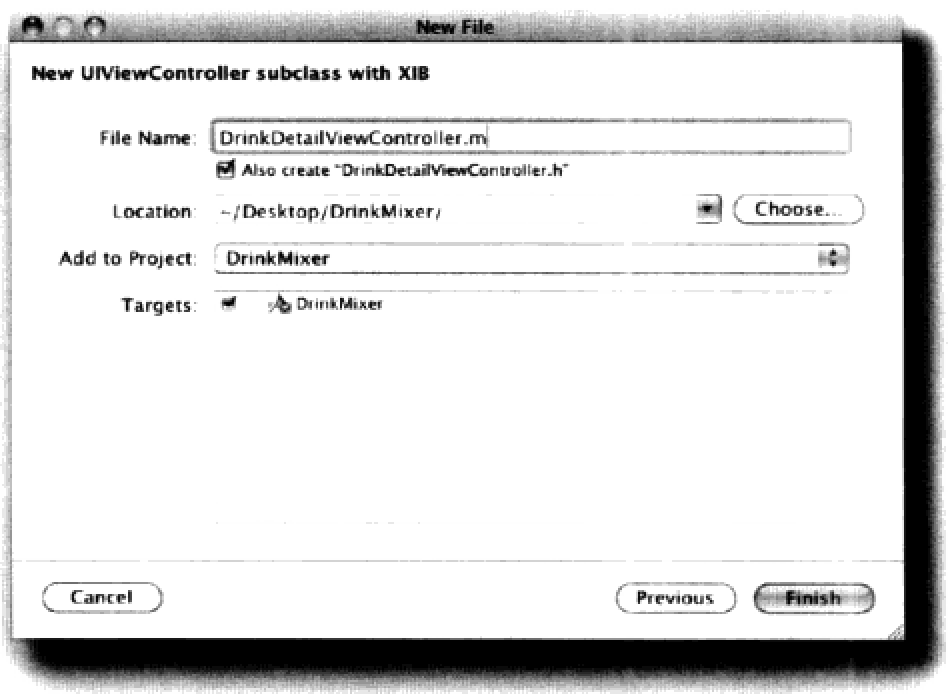


请一定要选中iPhone OS下面Cocoa Touch Class这一行。

选择UIViewContrller subclass这一项。

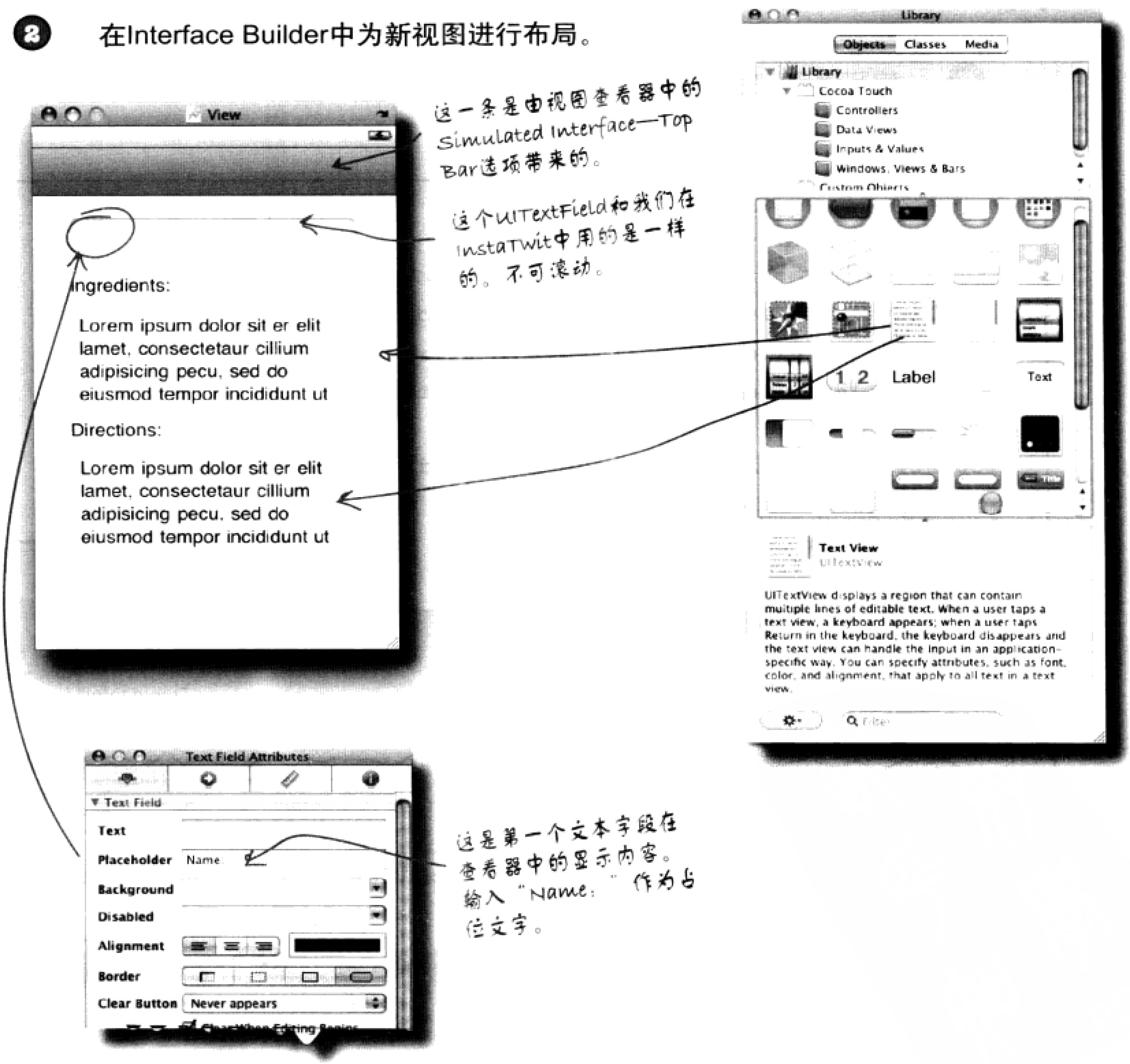
打开这个点选框，这也就会同时生成nib文件。

创建新文件对话框有许多选项，可以生成不同的新文件。对于我们来说，需要同时生成nib和相关支持文件，所以我们要创建一个新的Cocoa Touch类的.m和.h文件，以及.xib文件。



还有一件事。Xcode会在同一个目录下创建所有文件。要想让这些文件在Xcode中仍然按照你的方式进行组织，请将DrinkDetailViewController.xib文件移动到**/Resources**中。

2 在Interface Builder中为新视图进行布局。



【拼吾爱-最新编程资源的分享下载站】



这里是创建新的细节视图需要的所有信息。做完这个练习之后，你就应该得到一个可以工作的（但还是空白的）细节视图了。

3 **编写代码，完成新字段的声明和输出口定义。**

```
#import <UIKit/UIKit.h>
@interface DrinkDetailViewController : UIViewController {
        IBOutlet UITextField *nameTextField;
        IBOutlet UITextView *ingredientsTextView;
        IBOutlet UITextView *directionsTextView;
}
@property (nonatomic, retain) UITextField *nameTextField;
@property (nonatomic, retain) UITextView *ingredientsTextView;
@property (nonatomic, retain) UITextView *directionsTextView;
@end
```



DrinkDetailViewController.h

```
@implementation DrinkDetailViewController
@synthesize nameTextField, ingredientsTextView, directionsTextView;
```



DrinkDetailViewController.m

【拼吾爱-最新编程资源的分享下载站】

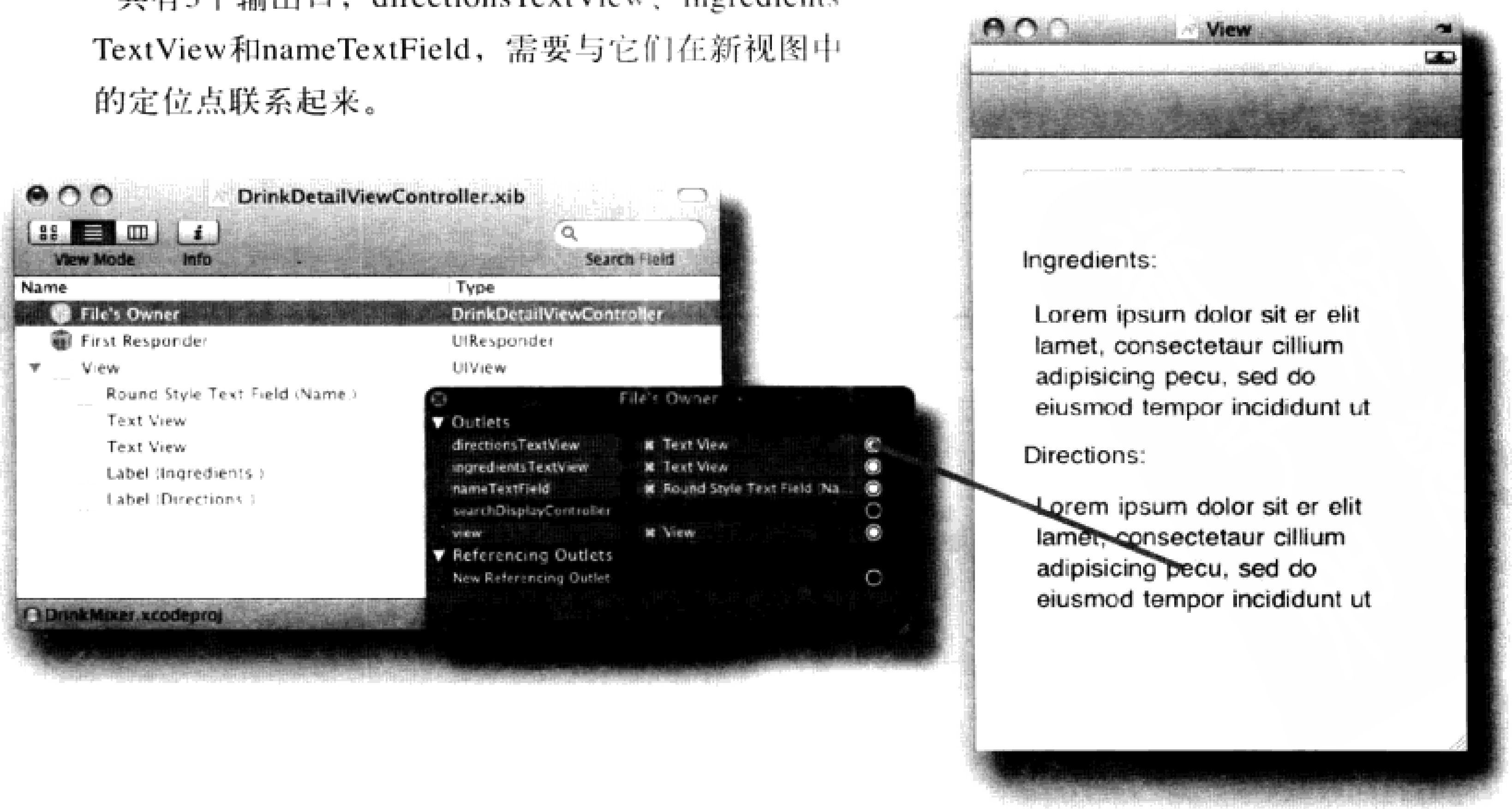

```
    -(void)
dealloc {
        [nameTextField release];
        [ingredientsTextView release];
        [directionsTextView release];
    [super dealloc];
}
@end
```



DrinkDetailViewController.m

4 **将细节视图与新的输出口连接起来。**

一共有3个输出口，directionsTextView、ingredientsTextView和nameTextField，需要与它们在新视图中的定位点联系起来。

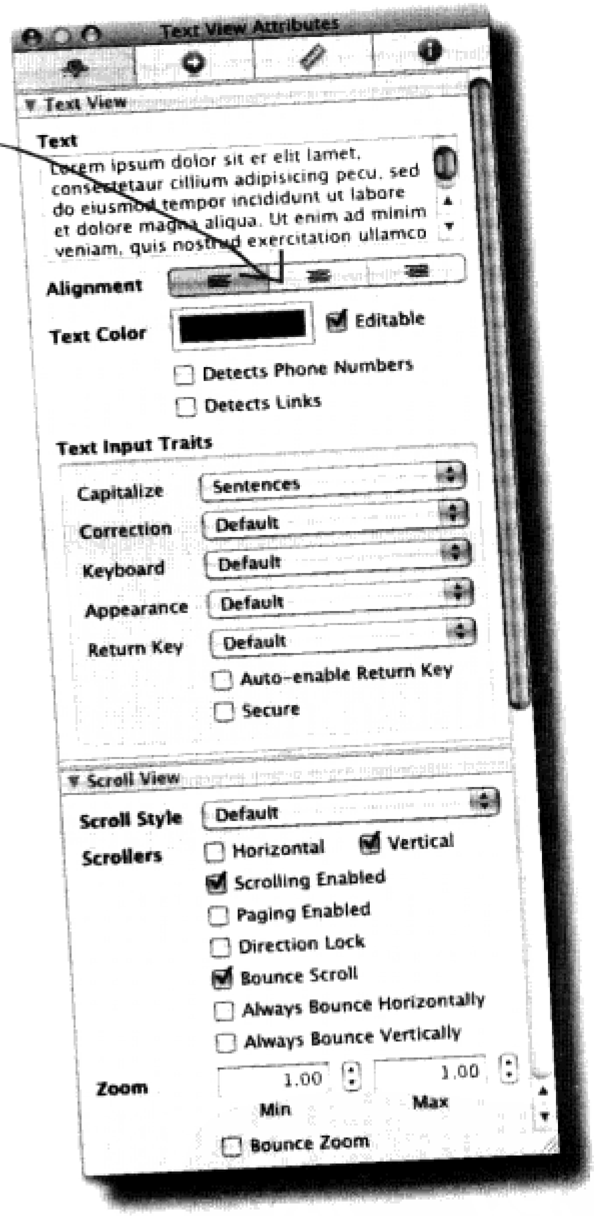


【拼吾爱-最新编程资源的分享下载站】



练习答案

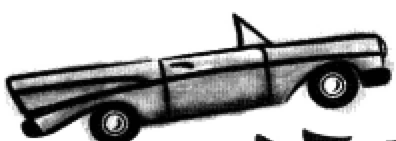
取消这个点选框的选中状态，可以冻结UITextViews的内容。



5 **将文本字段设为不可更改。**

我们需要把这个UITextField和2个UITextViews都设为禁用，以防止用户对其进行修改。只要单击每个字段，然后将这个点选框切换成关闭状态即可。

这些修改做完以后，键盘问题就解决了，因为不会出现键盘了！



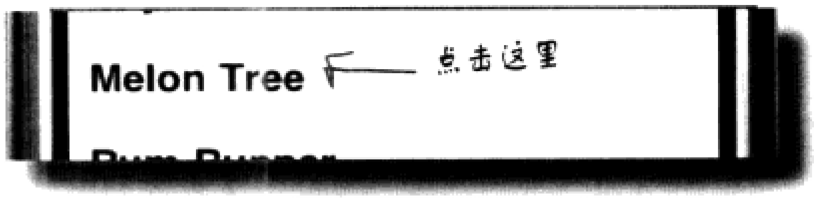
运行测试

构建并运行应用。你做了许多工作，现在是检查错误的时候了。

【拼吾爱-最新编程资源的分享下载站】

好吧，我这儿有人点了一个Melon Tree……但是我怎么还是看不到饮料的细节。



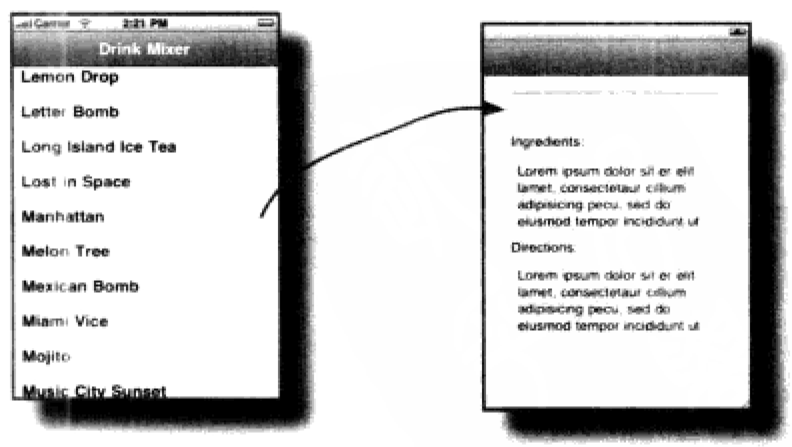


我们还是得在山姆选中一种饮料的时候去加载细节视图。



练习

当用户浏览饮料信息的时候，他们会在列表和细节视图之间来回切换。请考虑一下，我们怎样才能在切换的同时又不会让用户感到晕头转向。



1. 用户怎样在视图之间来回切换？

2. 我们怎样才能清楚地知道该显示哪一个视图？

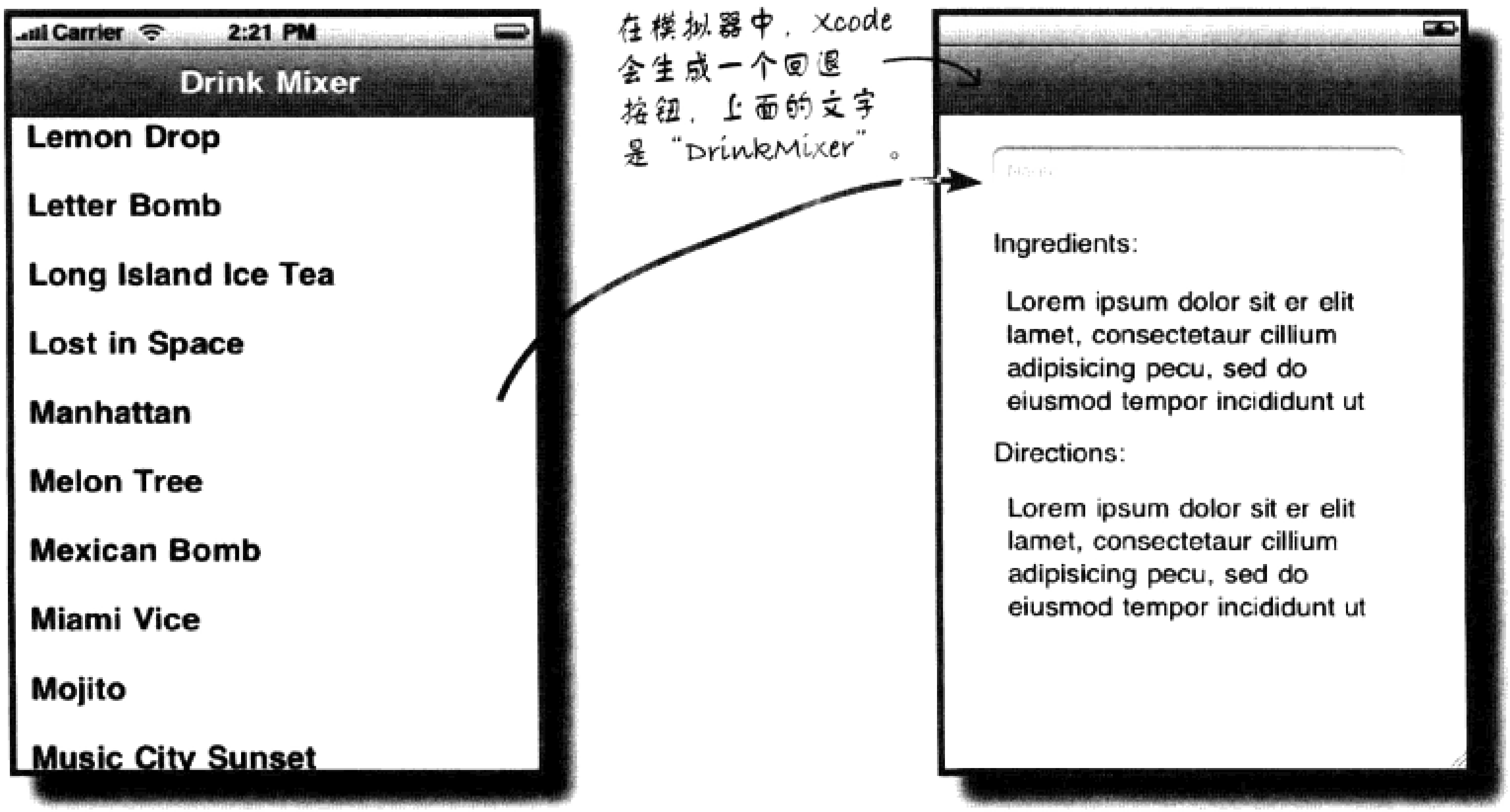
3. 细节视图怎样才能知道要显示哪种饮料？

4. 我们怎样才能让用户回到表格视图？

【拼吾爱-最新编程资源的分享下载站】

练习答案

当用户浏览饮料信息的时候，他们会在列表和细节视图之间来回切换。请考虑一下，我们怎样才能在切换的同时又不会让用户感到晕头转向。



1. 用户怎样在视图之间来回切换？用户想看哪一种饮料，就会点击显示有该饮料名称的单元格。

2. 我们怎样才能清楚地知道该显示哪一个视图？导航控制器会用回退按钮和导航栏标题来跟踪视图状态。

3. 细节视图怎样才能知道要显示哪种饮料？这可以根据用户选中的表格单元判断出来。

4. 我们怎样才能让用户回到表格视图？导航控制器会提供一个回退按钮，点击该按钮就可以回到主视图。

【拼吾爱-最新编程资源的分享下载站】

用导航控制器在视图之间切换

现在我们已经填充了表格视图的内容，也构造了细节视图，也是时候去考虑如何在两个视图之间来回移动了。基于导航的模板自带了我们需要的功能：

- **一个视图堆栈，以便在视图间移动**
  当用户来回移动时，你可以让导航控制器显示相应的视图。导航控制器会跟踪用户所处的视图，并提供回退按钮。

- **一个导航栏，上有按钮和标题**
  导航控制器会控制导航栏显示相应的按钮，按钮会与正在显示的视图互动，导航栏还会显示一个标题，好让用户了解当前处于哪个视图。

- **一个导航工具栏，显示视图相关的按钮**
  导航控制器可以在屏幕底部显示一个工具栏，用来显示专门为当前视图定制的按钮。

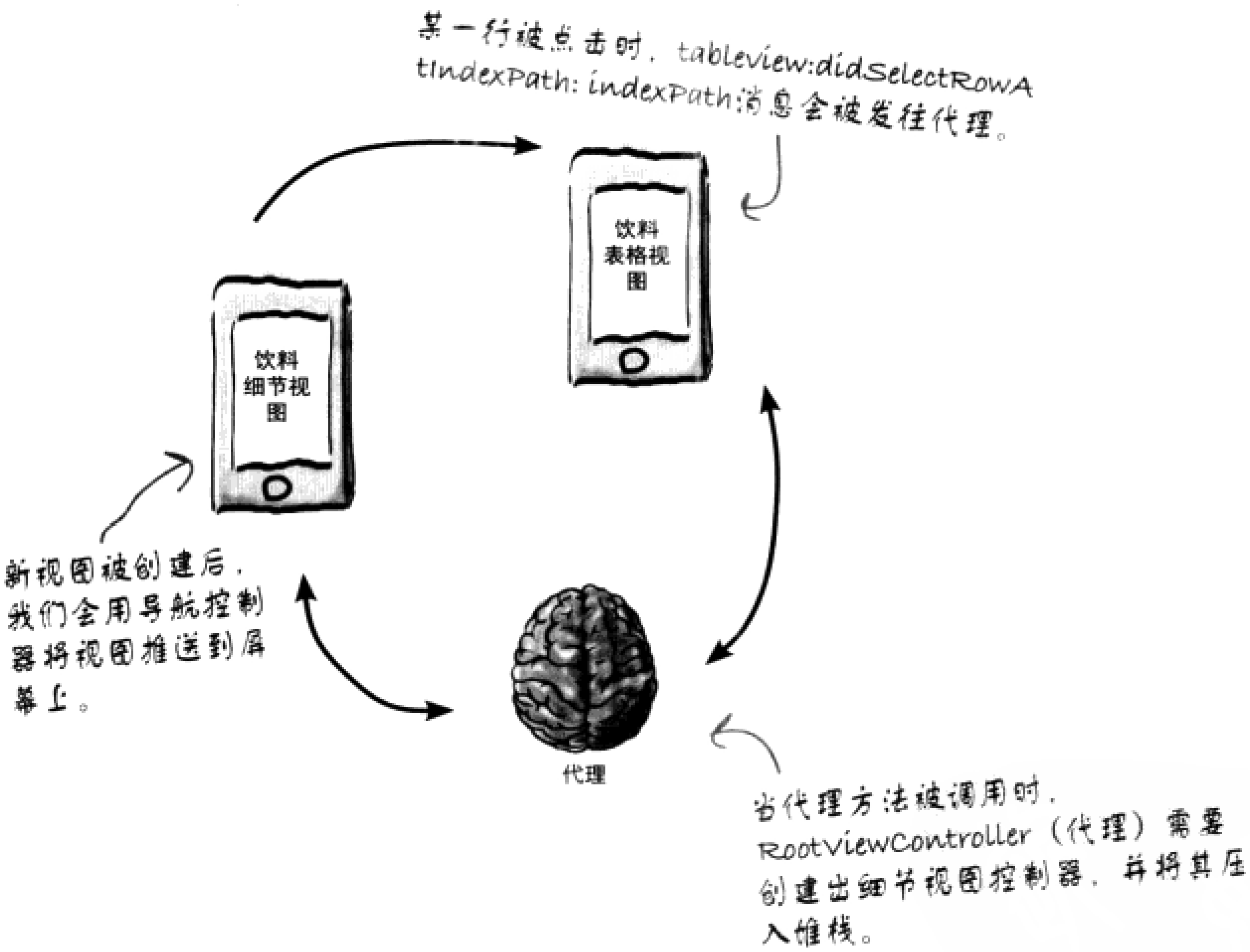
UINavigationController支持使用一个代理，名叫 UINavigationControllerDelegate，这个代理会在控制器将要切换视图时得到通知，但是对于DrinkMixer来说，我们不需要这一信息。视图本来就会在将要显示和隐藏的时候得到通知，对于我们的应用，有这一点就足够了。

现在我们需要让表格视图与导航控制器协同工作，来显示细节视图。

【拼吾爱-最新编程资源的分享下载站】

导航控制器维护了一个视图堆栈

从项目一开始，我们就把导航控制器拖了进来，现在终于能派上用场了。导航控制器会维护一个视图堆栈，并显示位于堆栈顶层的视图。它还会自动显示一个回退按钮，还有很酷的滑进滑出动画效果。在下一章中，我们会进一步完整地讲述导航控制器堆栈，但是目前要做的只是把新视图压入堆栈中，剩下的事情就交给控制器去操心了。我们只需要考虑怎么得到新视图就可以了。



我们会使用表格视图代理中的点击通知

当表格中的某一行被点击后，表格视图会调用代理的**tableview:didSelectRowAtIndexPath:**方法。表格会传递一个**NSIndexPath**（与**cellForRowAtIndexPath**类似）作为参数，告诉我们哪一行被选中了。

下面的事情就变得有趣了：我们的**RootViewController**就是代理，所以它需要把控制权交给细节视图的控制器……

【拼吾爱-最新编程资源的分享下载站】

实例化一个视图控制器与其他类没什么两样

剩下的最后一部分就是创建视图控制器了。实例化一个视图控制器与实例化其他某个类没什么两样，唯一的区别就是你可以把视图要加载的nib文件传给它作为参数：

```
[[DrinkDetailViewController alloc] initWithNibName:@"DrinkDetailView
Controller" bundle:nil];
```

一旦我们创建好细节视图控制器以后，就会请求NavigationController将新的视图控制器压入视图堆栈中。让我们把这些整合起来，在代理中编写回调方法，在其中创建新的视图控制器并压入堆栈：

```
#import "RootViewController.h"
#import "DrinkDetailViewController.h"
```

因为我们要创建新的视图控制器，所以需要包含它的头文件。

```
// Override to support row selection in the table view.
  - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
*)indexPath {

    // Navigation logic may go here -- for example, create and push another view
controller.
  DrinkDetailViewController *drinkDetailViewController =
[[DrinkDetailViewController alloc] initWithNibName:@"DrinkDetailViewController"
bundle:nil];
  [self.navigationController pushViewController:drinkDetailViewController
animated:YES];
  [drinkDetailViewController release];
}
```

这是代理的回调方法-indexPath会告诉我们哪一行(饮料)被选中。

实例化控制器…

…然后压入导航堆栈。

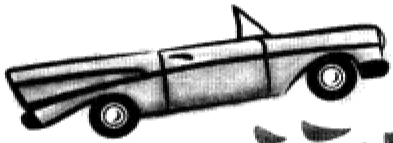
现在导航控制器已经持有了细节视图，我们可以释放对它的引用了。



RootViewController.m

让我们试一下吧……

【拼吾爱-最新编程资源的分享下载站】



运行测试

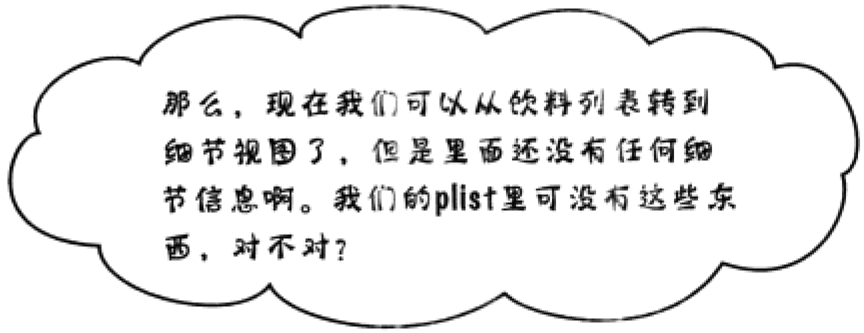
现在两个视图可以彼此沟通了，继续前进，构建、运行。

点这里就会让细节视图出现。



试着点击文本字段—键盘不会出现，因为这些字段不可编辑！

【拼吾爱-最新编程资源的分享下载站】





我们的数组有点膨胀过度了

现在剩下的事，就是把饮料成份和调制指南放进细节视图里去了，之后我们就可以有一个随身的酒保参谋了。为了省去你输入这些成份和指南的功夫，我们把额外的信息都放进一个新文件中去。问题是我们不能简单地把这些信息塞进一个数组里。要在这个版本中加入饮料的细节，我们需要一种不同的数据模型。



练习

下列选项中，哪些可以用来加载饮料数据？

- ☐ 用饮料信息创建一个数据库
- ☐ 用我们的plist中的字典来装载饮料细节
- ☐ 用一个XML文件装载饮料细节
- ☐ 在我们的plist中创建多个数组

这些选项中哪一个最适合DrinkMixer？为什么？ ....

....

....

....

【拼吾爱-最新编程资源的分享下载站】

练习答案

下列选项中，哪些可以用来加载饮料数据？

也可以用数据库来存储饮料信息，但是由于这个应用里其他地方都没用到数据库，所以得多做一些工作才能支持DB…再看看吧。

我们已经有一个存字符串的plist了——转换到存字典的plist不会太麻烦，还能得到可以保存饮料信息的数据结构。

- [ ] 用饮料信息创建一个数据库
- [x] 用我们的plist中的字典来装载饮料细节
- [ ] 用一个XML文件装载饮料细节
- [ ] 在我们的plist中创建多个数组

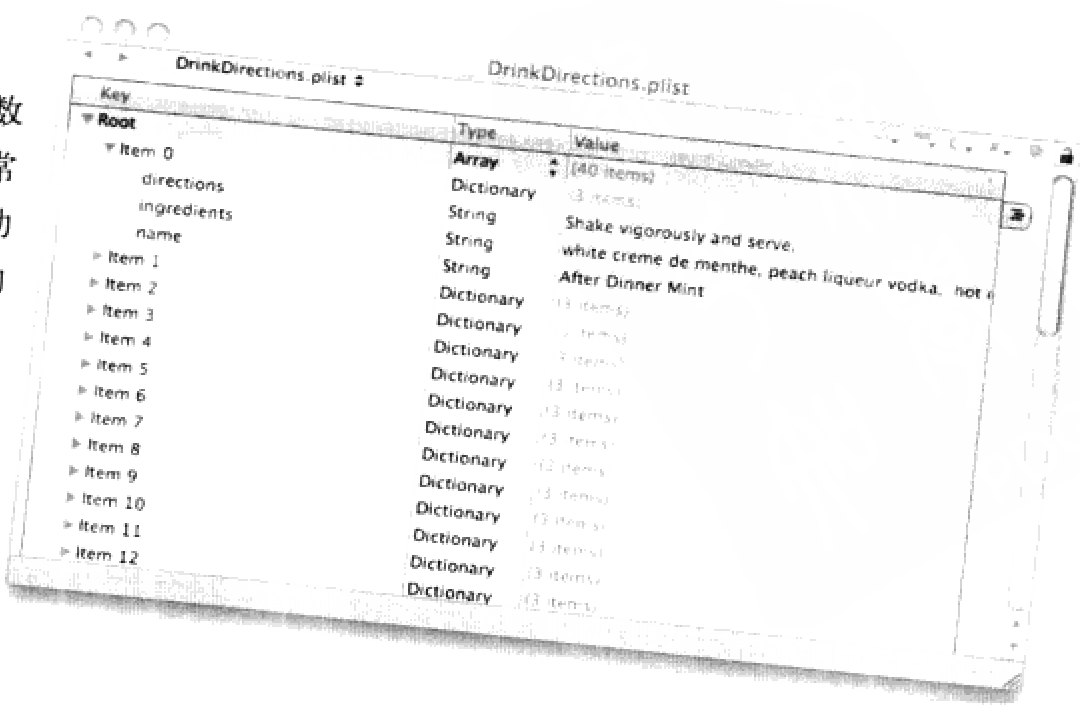
这也行，但和使用DB一样有一些负担。现在我们没有解析过XML，所以必须定义schema，然后再添加解析代码。

基本上这是所有选项中最差的－我们必须保证让多个数组排列整齐，才能正确保存一种饮料。

这些选项中哪一个最适合DrinkMixer？为什么？因为我们已经用到了plist，所以可以把它从一个字符串数组转换成字典的数组，又不用费太大的力气。这样我们也就不必在项目中引入SQL或者是XML了。不过，我们也失去了SQL和XML可以带来的强类型和数据检查。由于这是一个较小型的项目，我们打算就使用字典了。

字典将信息按照键－值对来存储

我们当前的饮料plist仅仅是一个饮料名称的数组。对于向表格视图填充饮料名称来说非常好用，但是对于饮料细节方面就没什么帮助了。对于这个plist，我们会创建一个字典的数组，而不是原来的字符串数组。在每个字典中都会有3个键：`name`、`ingredients`和`directions`。每个键都有字符串类型的值，里面存有相应的信息。由于NSDictionary遵循NSCoding协议，所以它可以在plist中保存并加载，就像我们原来的数组一样。



【拼吾爱-最新编程资源的分享下载站】

有问必答

问： 你一直在说什么NSCoding。这是什么啊？

答： NSCoding是一种协议，用于对象的编码与解码。使用这个协议就意味着要处理如何将一个对象存储到磁盘上或者在设备间分发。要了解更多关于NSCoding的信息，请参看苹果公司的文档。

问： 细节视图上面的回退按钮是从哪里来的？我们没做过什么啊……

答： 这是导航控制器自带的一种自动功能。当你为主视图添加了标题之后，导航控制器就会将这个名字当作导航用的视图堆栈的一部分，并且会添加一个回退按钮，将标题写在上面。所以你说对了，不是你干的！

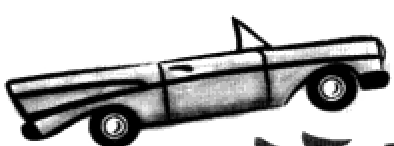


即食代码

回到http://www.headfirstlabs.com/iphonedev这个网址，下载DrinksDirections.plist。这次的文件名字不一样，所以你得进行一两处简单的修改。

1. 在Xcode中打开新的plist文件（还是放在resource目录下），看一看里面都有什么——所有数据都准备好了，随时待命！

2. 进入代码中，将对DrinkArray的引用改为对DrinksDirections的。



运行测试

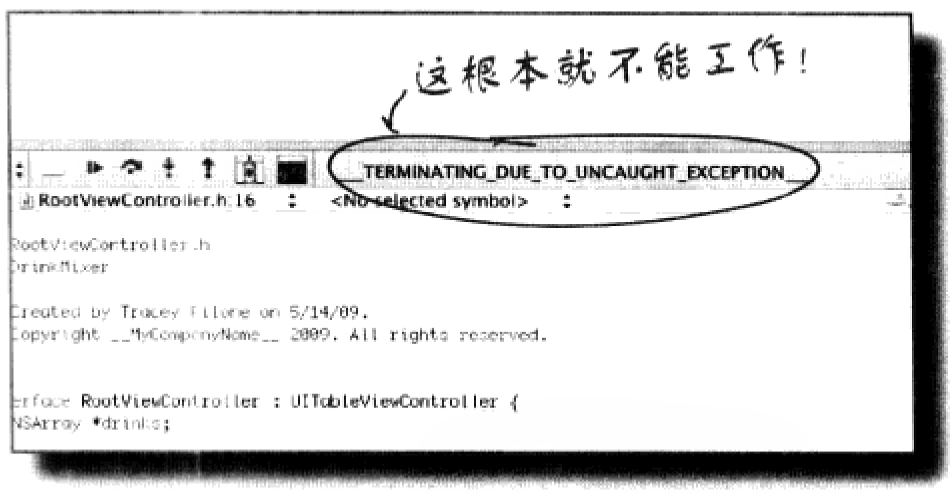
用新的plist构建并运行应用，看看会发生什么……

【拼吾爱-最新编程资源的分享下载站】

运行测试







【拼吾爱-最新编程资源的分享下载站】

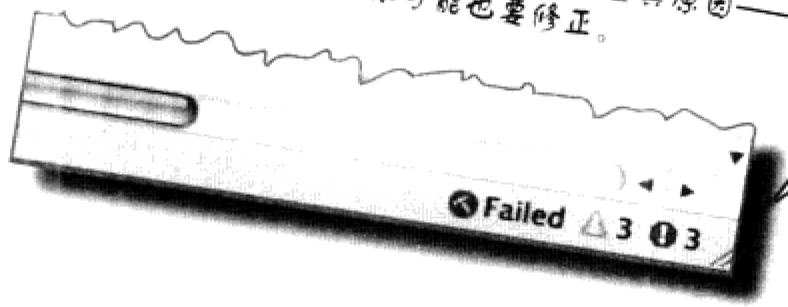
调试——iPhone开发之暗面

事情有些不对劲，不过老实说，这种事在开发过程中再正常不过了。有很多因素都可能会导致我们的应用程序崩溃，所以需要找出问题所在。

警告信息能帮我们不用调试就找出问题

一般情况下，如果你的应用程序无法编译构建成功，Xcode根本就不会启动它——但是对于警告信息就并非如此了。Xcode会很正常地编译并运行一个有警告的应用程序，你仅仅会在Xcode的右下角看到一个黄色的惊叹号。花几分钟时间检查一下出现警告的原因，就可能为你省下以后几个小时的调试时间。

3个错误和3个警告…错误必须被修正。警告也应该检查其原因——而且很可能也要修正。





极客点滴

一些常见的警告肇因：

1. 自从iPhone OS 3.0发布以后，使用了将要作废的2.0属性的代码会引发警告。

2. 向一个对象发送消息，而该对象并未声明支持该消息（由笔误或者自动补完代码出错引起的）也会引发警告。你的应用可以编译，但是很可能会在执行这段代码时出现运行时异常而终结。

不过，我们的问题并不在于此：我们的代码应该是完全没有警告和编译错误的。好的一面是，当应用在模拟器中崩溃的时候，并不会直接消失（在真实设备上会如此）。Xcode会在系统正常关闭应用之前将其停止。让我们利用这一功能，来看看到底是怎么回事。

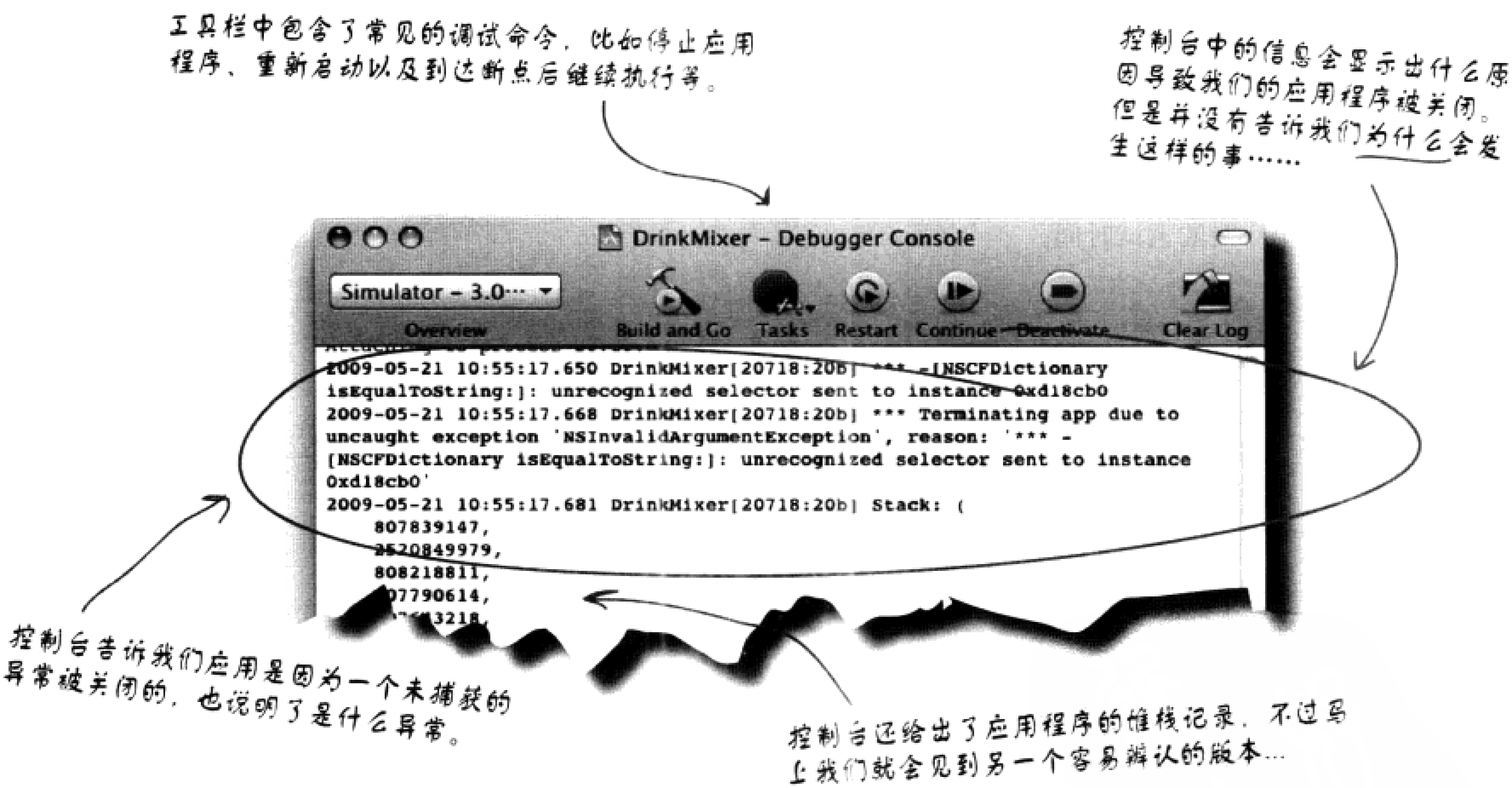
是时候做一些调试了……

【拼吾爱-最新编程资源的分享下载站】

调试冒险之旅第一站：控制台

我们需要找出应用崩溃的原因，而很幸运的是，Xcode具有很多强有力的调试功能。目前我们只是打算看一下它提供的崩溃信息，但是以后在本书中我们还会进一步介绍一些更高级的调试功能。

由于你是在模拟器中运行程序的，控制台应该已经开启。我们的控制台看起来应该就是下面这个样子：



轻松一刻

如果你没见到控制台，我们可以把它给你弄出来！

如果你是在另一种状态下运行DrinkMixer的，或者看不到控制台，可以在Xcode里打开**Run → Console**菜单选项。

【拼吾爱-最新编程资源的分享下载站】

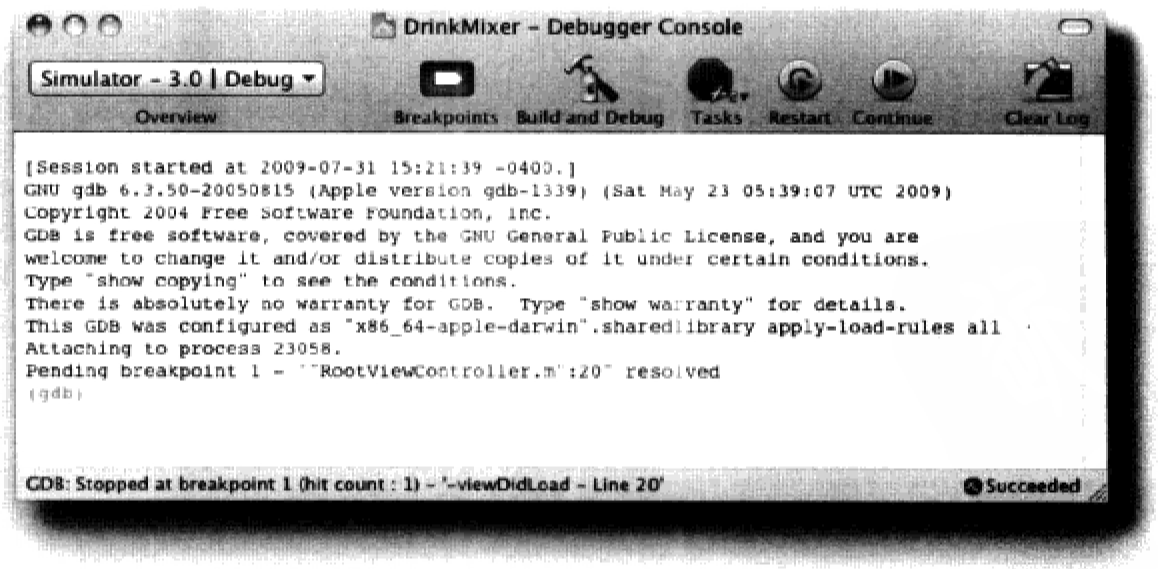
与你的应用程序在运行期间互动

控制台是一件非常强力的调试工具。一些最佳的调试技术都涉及在合适的位置上用NSLog(...)来输出日志消息。这些信息都会打印到控制台上，可以帮助你快速诊断问题所在。不过控制台并不是只读的，它是你深入运行中的应用程序的窗口。我们会在控制台中看到日志消息，而当应用程序到达一个断点的时候，控制台中会出现一个命令行提示符。在这个命令行里你可以使用各种调试命令，比如**print**、**continue**、**where**、**up**和**down**等，用这些命令可以查看应用程序的状态。

控制台调试器实际上就是开源软件gdb，所以几乎所有的gdb命令都可以在里面使用。

而当应用将要停止运行的时候

在这种情况下，我们是在处理一个几乎要死掉的应用程序，但是理念是相同的。由于DrinkMixer已经崩溃了，Xcode会提供关于出错的基本信息。对于我们而言，错误就是向一个对象发送了“无法识别的选择子”。记得吗？选择子其实差不多就是一个方法调用——也就是说，有代码在试图调用一个对象的方法，而这些方法并不存在。



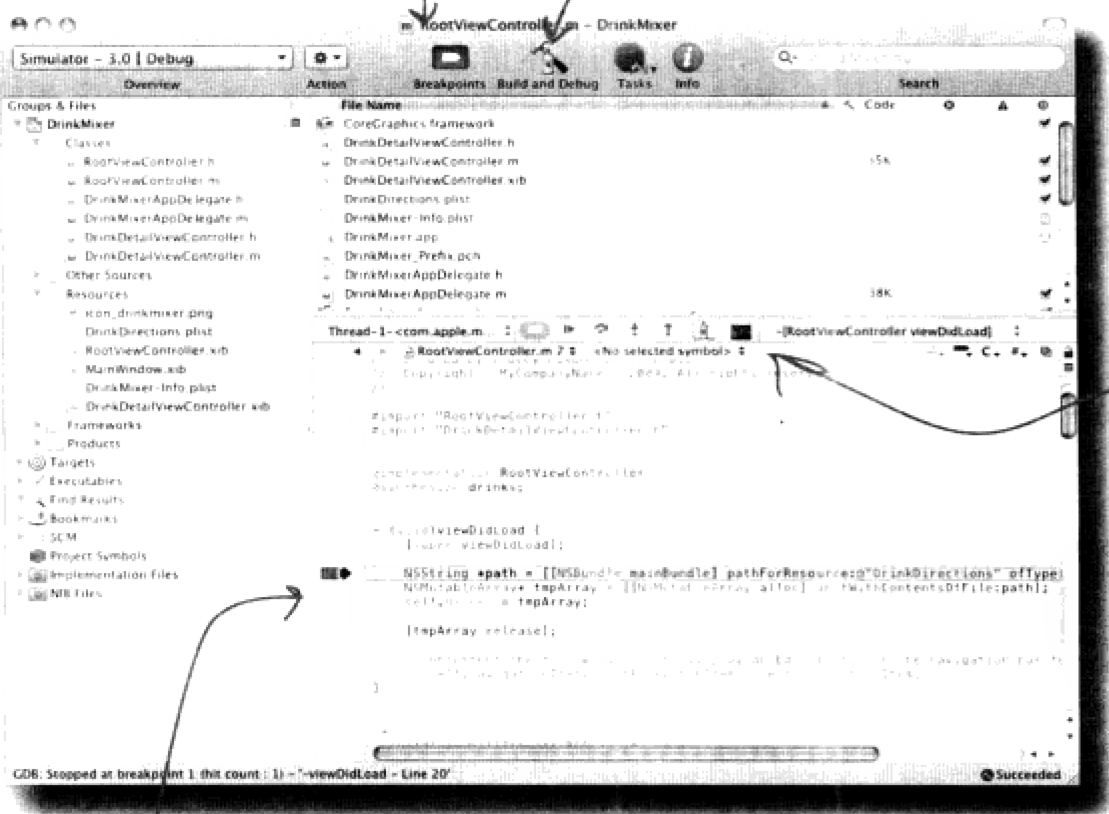
控制台提示符让你可以与应用程序通过命令行互动。

但是Xcode并不仅仅提供了命令行。它还自带有一个完整的GUI调试器。让我们看一看……

【拼吾爱-最新编程资源的分享下载站】

Xcode在应用中断运行后也会为你提供支持

到目前为止，我们已经用Xcode写过代码，编译启动过应用程序。但是它的用途在我们点击“Build and Debug”按钮以后会进一步体现出来。首先，我们可以在代码中设置断点，好让我们能了解运行状态。只要简单地单击想要停下的代码行旁边的地方，就可以设置断点了。Xcode会在这一行边上加上一个小小的蓝色箭头，当你的应用程序运行到这一行代码的时候，就会停下来，让你可以用控制台看看情况。



单击小小的杀虫剂图标，或者按下Shift-⌘-Y键，就可以启动调试器……

【拼吾爱-最新编程资源的分享下载站】

Xcode调试器会向你展示应用程序的状态

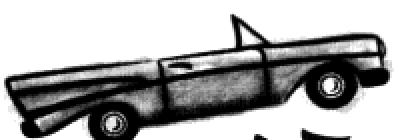
调试器会展示出你的代码以及堆栈，还有一个窗口可以看到变量和内存。当你单击堆栈的某一层时，Xcode会显示与之相关联的那一行代码以及相应的局部变量。你在调试器窗口中所做的一切，都可以在控制台中做到，但是调试器在控制台之上提供了一个不错的图形界面。

这里是单步跟踪和继续执行按钮，可以逐步运行代码。



这是应用在当前断点（或者崩溃时）的堆栈。如果你点击堆栈某一层，Xcode会显示出相应的代码。

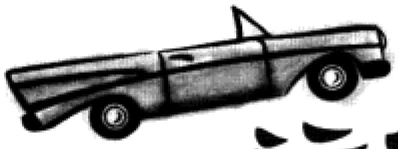
Xcode会在这个视图中显示应用的变量（本地变量、全局变量等）。



运行测试

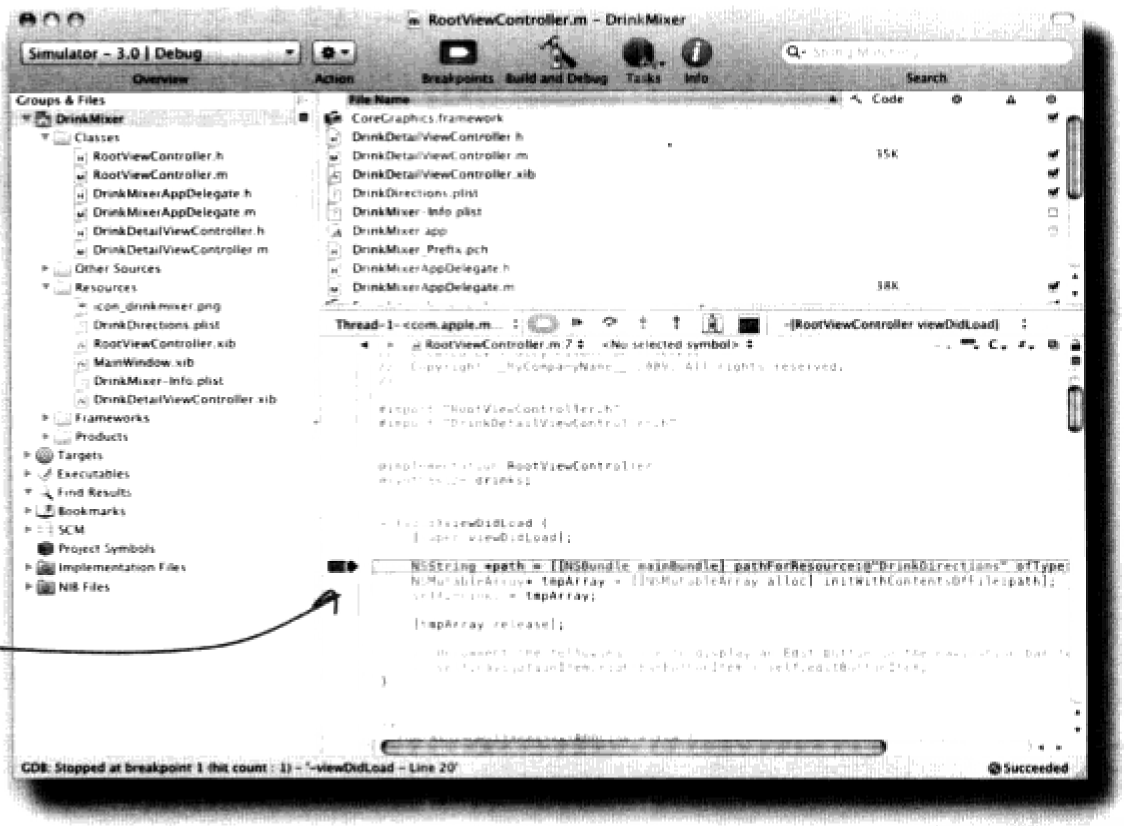
因为我们知道问题大概出在数组上，所以请试着在那里设置一个断点。然后构建并运行应用，看看会发生什么。

【拼吾爱-最新编程资源的分享下载站】

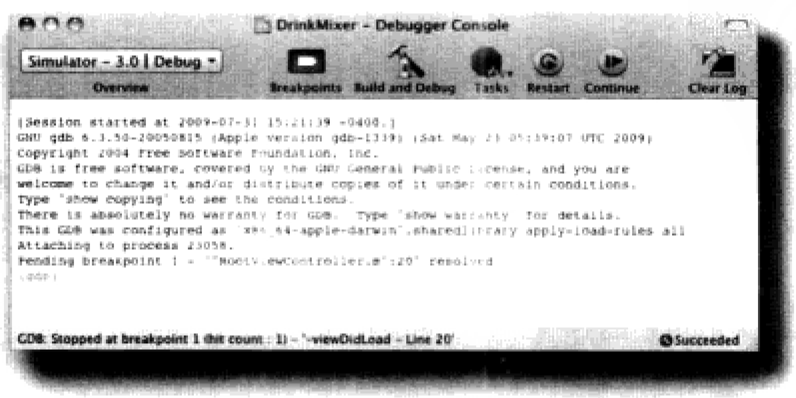


运行测试

当你在加载数组的地方设置好断点后，运行应用，没有出现任何问题。



要设置断点，需要在这里的空白处点击。



但是点击继续执行时……

【拼吾爱-最新编程资源的分享下载站】



这到底是怎么回事?

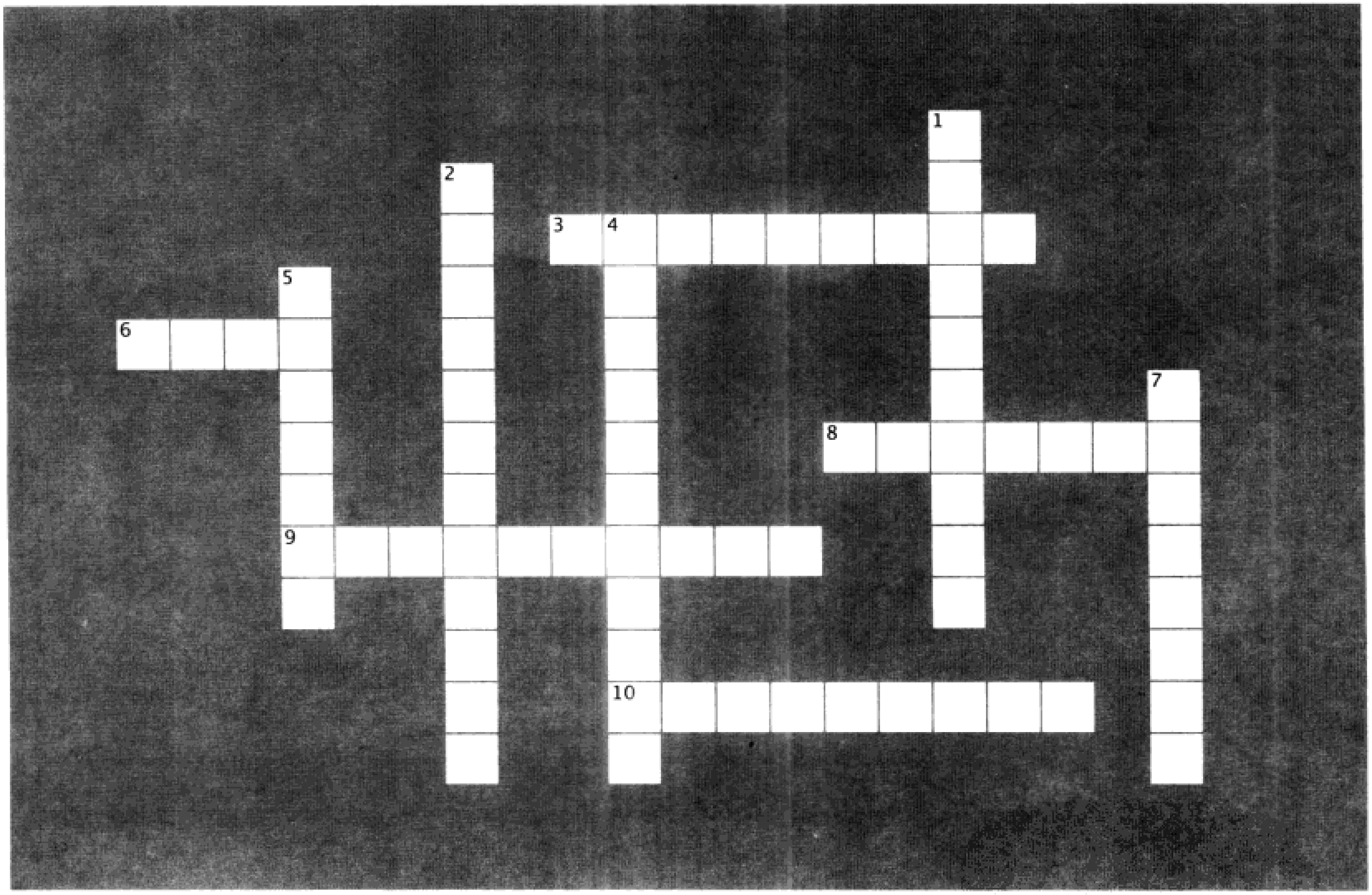
我们的应用程序一直崩溃，可又不是在加载数组的代码里出的问题。打开调试器，单击堆栈含有我们代码的最上面一层。你会看到引起崩溃的代码行……发现哪里错了吗?

待续……

【拼吾爱-最新编程资源的分享下载站】

多视图纵横字谜

用你学到的关于导航控制器和多视图的字眼，填入下面的空格中。



横向提示

3. 一组视图，由导航控制器进行操纵。
6. 字典使用__________来组织数据。
8. 用于显示应用输出的屏幕画面。
9. 一种组合了表格视图与导航控制器的模板。
10. 拥有一些单元格，要进行定制才能用。

纵向提示

1. 数组之外一种更灵活的管理数据的方式。
2. DrinkMixer就是这种类型的应用。
4. 要使用一个新的类，你就得________它。
5. 符号@是创建这种对象的一种简便方式。
7. Xcode中的工具，可以修复代码中的错误。

【拼吾爱-最新编程资源的分享下载站】



你的iPhone工具箱

你已经消化吸收了第4章，现在可以把多视图与导航控制器加入到你的工具箱中了。如果你想得到一份完整的书中功能提示的列表，请访问http://www.headfirstlabs.com/iphonedev。

导航模板：

自带有一个内建的表格视图与导航控制器。

非常适合生产力应用。

设计用于管理分层数据和多视图。

内建视图间切换的动画效果。

表格：

是一组单元格的集合。

支持编辑内容、滚动以及移动表格行。

可以进行定制，让单元格看起来不止一列。

UITableView：

通过仅创建视图中出现的单元格来控制内存使用。iPhone需要为其他对象分配空间时，任何不在视图中的单元格都会被销毁。

导航控制器：

维护一个视图堆栈，以便在视图间导航。

有一个导航栏，可以放置按钮和标题。

可以支持按需要在视图底部显示自定义工具栏。

Plist：

Xcode和数组都支持plist。

一种存储信息的良好方式。

善于处理数据，但有一定局限——我们会在下面两章里介绍另一种方法Core Data。

调试：

有内建的控制台，可以进行调试与日志信息输出。

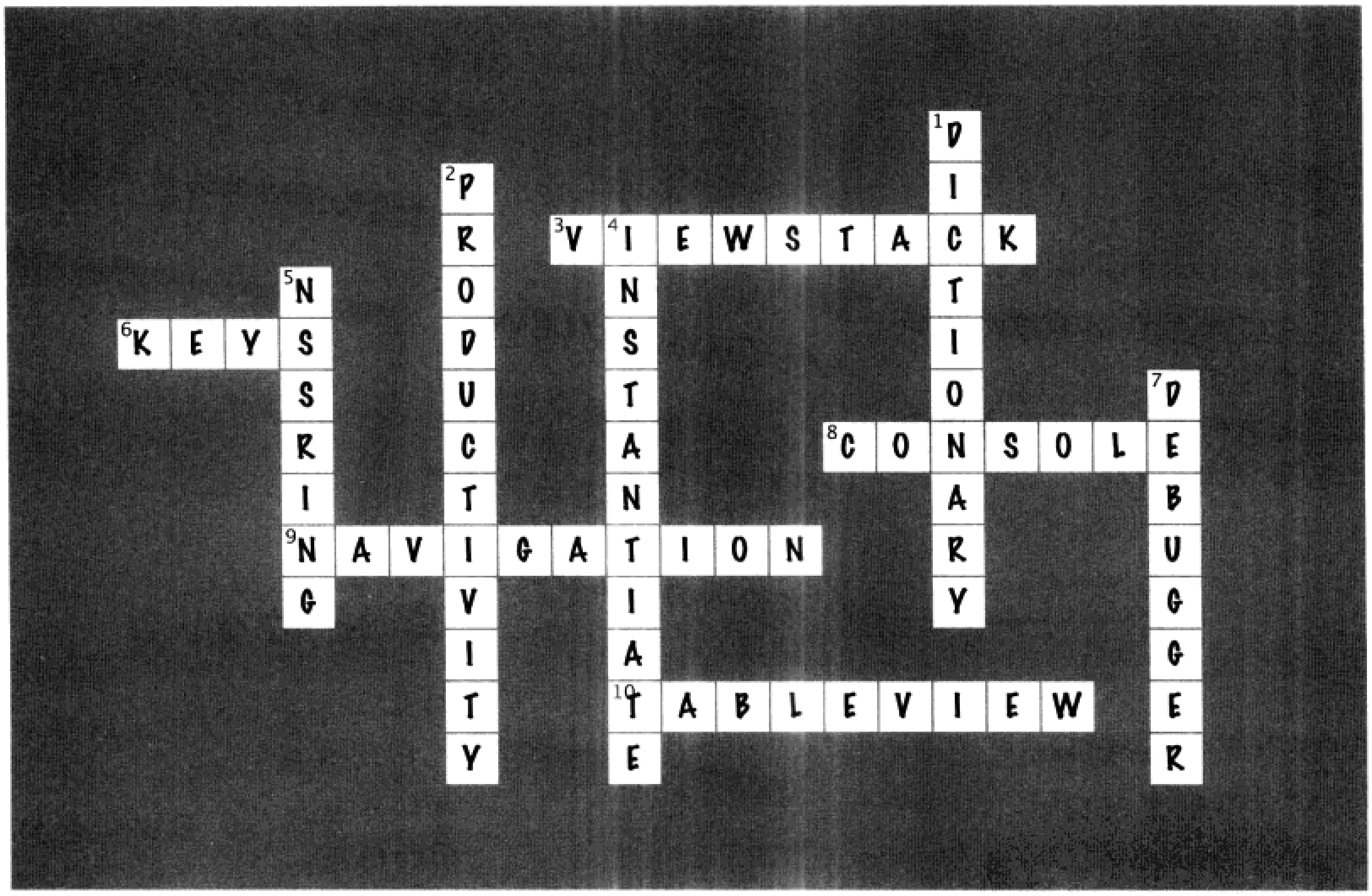
在编译时给出错误与警告信息，以定位问题。

有内建的调试器，让你可以设置断点并单步执行代码，来查找错误。

【拼吾爱-最新编程资源的分享下载站】

多视图纵横字谜答案

用你学到的关于导航控制器和多视图的字眼，填入下面的空格中。



横向提示

3. 一组视图，由导航控制器进行操纵。[VIEWSTACK]
6. 字典使用_________来组织数据。 [KEYS]
8. 用于显示应用输出的屏幕画面。 [CONSOLE]
9. 一种组合了表格视图与导航控制器的模板。[NAVIGATION]
10. 拥有一些单元格，要进行定制才能用。[TABLEVIEW]

纵向提示

1. 数组之外一种更灵活的管理数据的方式。[DICTIONARY]
2. DrinkMixer就是这种类型的应用。[PRODUCTIVITY]
4. 要使用一个新的类，你就得_______它。[INSTANTIATE]
5. 符号@是创建这种对象的一种简便方式。[NSSRING]
7. Xcode中的工具，可以修复代码中的错误。[DEBUGGER]

【拼吾爱-最新编程资源的分享下载站】

5 Plist与模式视图

对应用千锤百炼



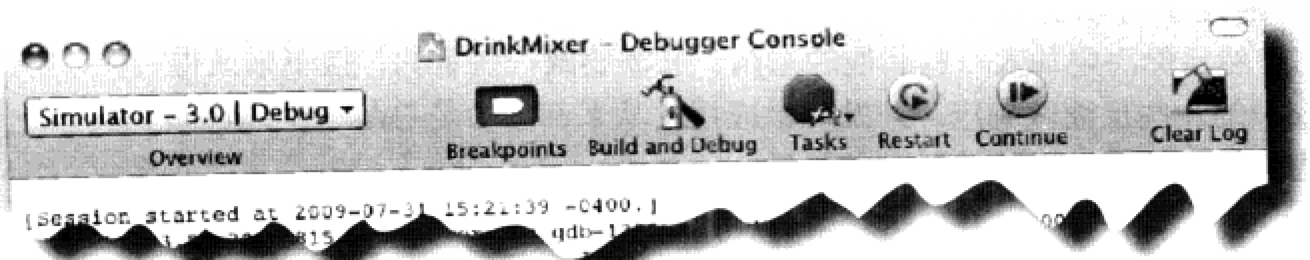
那么你的应用几乎可以工作了……每个应用都会经过这一步！首先会有一些功能可以使用，然后你决定加入一些其他的东西，又需要做一些重构，接下来又得对App Store中的一些反馈做响应。开发一个应用，从来都不会是~~总是~~一个线性的过程，但是在这个过程中可以学到许多东西。

【拼吾爱-最新编程资源的分享下载站】

一切要从山姆说起……

山姆需要一个应用，来帮助他更轻松地完成酒保的工作。你很快就弄了一个出来，但是在填充每种饮料的细节时碰到了点麻烦，因为用到了一个里面装有字典的plist。





我们上次离开DrinkMixer的时候，它正处于调试当中……

【拼吾爱-最新编程资源的分享下载站】

崩溃的解剖报告

DrinkMixer启动并运行正常，一直到达我们在第20行设置的断点为止。调试器会将应用程序停下来，并显示出调试控制台。通过在代码中设置断点，我们在第4章结尾处发现，在应用开始执行到导入文件的命令之前，都没有发生崩溃；到目前为止一切正常。

让我们逐步执行加载plist的过程。为了确保正确执行，我们要输入两次**next**。第一个“next”是为了查找plist的路径，第二个会实际加载数据。



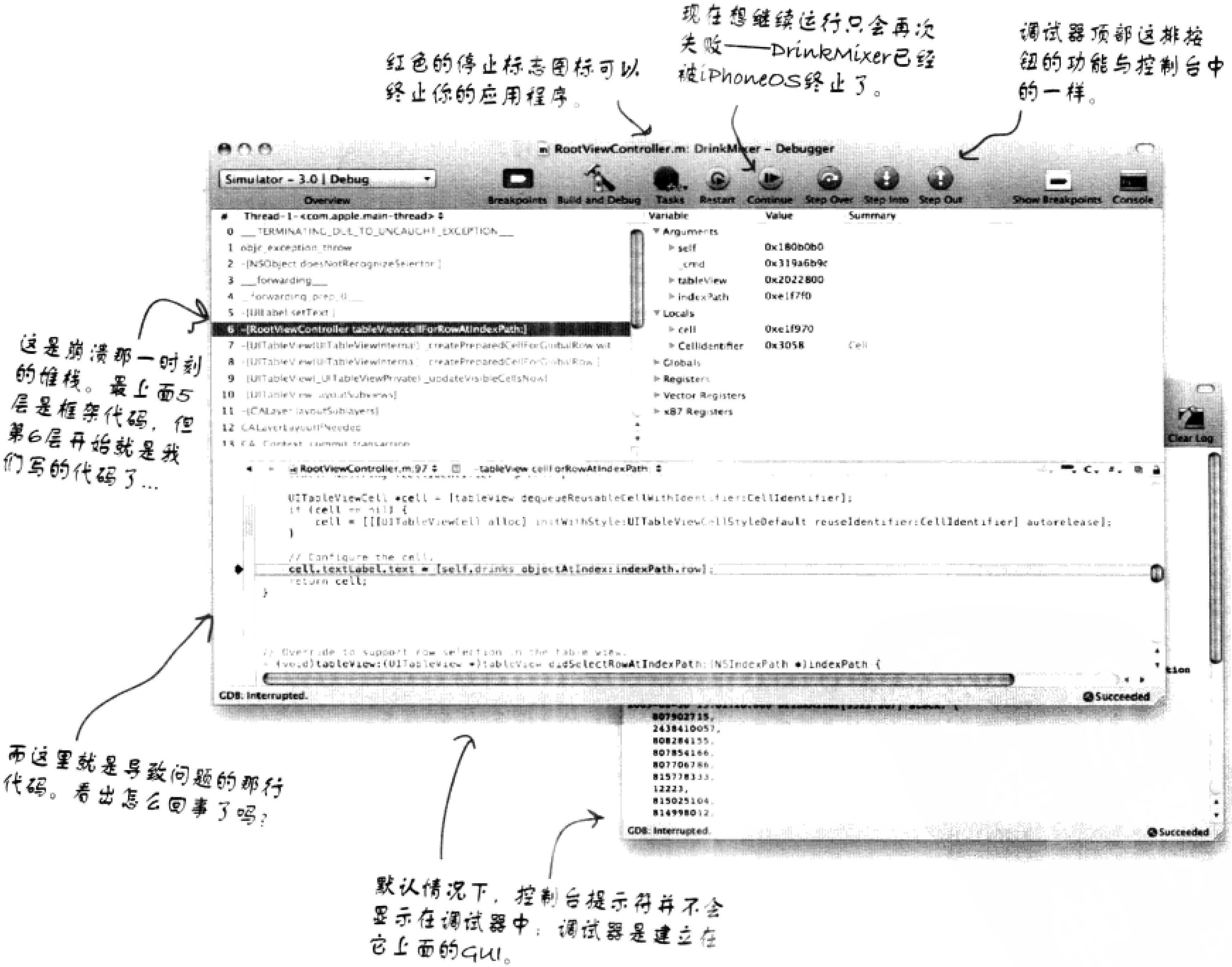
加载plist工作正常，没有任何问题。错误一定是发生在那以后。我们让应用继续运行，看看到哪里出了错。按下“继续运行”按钮（或者在控制台中输入**continue**）……然后异常就又出现了。这个错误到底是在哪里发生的呢？

【拼吾爱-最新编程资源的分享下载站】

用调试器调查崩溃的罪魁祸首

我们可以复现DrinkMixer的崩溃，而看起来又不是由我们的plist加载代码引起的。在iPhoneOS将我们的应用程序关闭之前，Xcode已经将其暂停，这样我们就可以用调试器来找出在崩溃之前它到底想要做什么。

切换回调试器，看一下左上角显示的堆栈。这就是导致了崩溃的调用堆栈。



【拼吾爱-最新编程资源的分享下载站】

拿起笔来

运用你目前所学过的知识，找出到底是怎么回事！

这个异常是关于NSCFDictionary的。与之相对应的到底是哪个字典？它是从哪儿来的？

是谁在向这个字典发送消息？为什么我们会收到一个无法识别的选择子？

【拼吾爱-最新编程资源的分享下载站】

拿起笔来

运用你目前所学过的知识，找出到底是怎么回事！

这个异常是关于NSCFDictionary的。与之相对应的到底是哪个字典？它是从哪儿来的？

这个字典是来自plist的！当我们加载了这个plist以后，就拥有了一个字典的数组，而不再是字符串数组了。

是谁在向这个字典发送消息？为什么我们会收到一个无法识别的选择子？

消息是在设置单元格标签文字时发往字典的。实际上是标签向它发送了这个消息（看看下一层堆栈，代码是在UILabel中）。以前发送这个消息时假定标签文字是字符串，但现在我们是在把一个字典赋给标签文字。

```
    // Configure the cell.
    cell.textLabel.text = [self.drinks objectAtIndex:indexPath.row];
    return cell;
}
```

我们正在试图把一个字典塞进字符串里

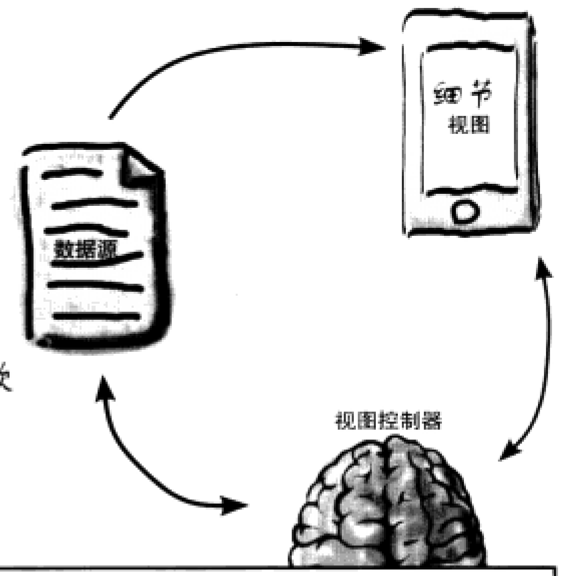
把一个字典放进标签的文字中，而后者要的是一个字符串，肯定无法成功。我们以前的数组是一个字符串数组，所以这段代码工作得很好。现在我们换成了一个字典的数组，就得想办法从里面取出饮料名称的值（字符串），再把这个值赋给文本标签。如果你再仔细看看DrinkDirections.plist就会发现，里面有一个字典的数组——每种饮料一个字典。字典中是按照键来存储值的；它们只是一个键－值对的群集。要从里面将值取出来很简单，只要向字典发送`objectForKey:@"key"`消息即可。



【拼吾爱-最新编程资源的分享下载站】

让代码处理装有字典的plist

我们已经掌握了字典的组合方式，也就可以根据这些信息填充细节视图了。如果你将选中的饮料对应的字典交给细节视图控制器，它就可以在向用户显示视图之前，填充视图的各个域。



每个字典中都有要了解一种饮料所需的所有内容。我们要做的就是将这个字典交给细节视图。

拿起笔来

动手对你的应用进行修改吧。经此之后，它就应该知道你使用的是一个字典而不是字符串的数组了——而细节视图应该保留一个对它将要显示的饮料的引用。最后，细节视图应当在显示到屏幕上之前，为自己的域填充内容。

1. **改变表格单元的配置方式。**
   在RootViewController.m中，修改对单元格的textLabel.text属性进行赋值的方式，要使用来自相应的字典中的名称值。

   如果你想要了解更多关于字典的内容，不要忘记去查看NSDictionary的文档。

2. **在细节视图中添加一个对饮料字典的引用。**
   在DrinkDetailViewController.h中，加入一个NSDictionary*成员变量，命名为drink，并声明相应的属性。

3. **将drink添加到DrinkDetailViewController.m文件中。**
   为新的字典引用添加属性合成和内存释放代码。

   马上我们就会修改细节视图控制器，以使用新的字典中的值…

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

仔细检查一遍代码，确保一切正常……

```
// Configure the cell.
     cell.textLabel.text = [[self.drinks objectAtIndex:indexPath.row]
objectForKey:@"name"];
   return cell;
```

用objectForKey来从字典中获取名称。

RootViewController.m

```
@interface DrinkDetailViewController : UIViewController {
     NSDictionary *drink;
     IBOutlet UITextField *nameTextField;
     IBOutlet UITextView *ingredientsTextView;
     IBOutlet UITextView *directionsTextView;
}

@property (nonatomic, retain) NSDictionary *drink;
@property (nonatomic, retain) UITextField *nameTextField;
```

声明NSDictionary*类型的成员变量和属性，属性照常设置非原子操作并在赋值时持有对象。

DrinkDetailViewController.h

```
@implementation DrinkDetailViewController
@synthesize drink, nameTextField, ingredientsTextView,
directionsTextView;
```

将drink加入合成代码这一行之中。

```
 - (void)dealloc {
     [nameTextField release];
     [ingredientsTextView release];
     [directionsTextView release];
     [drink release];
     [super dealloc];
}
@end
```

在此释放我们的字典引用。

DrinkDetailViewController.m

【拼吾爱-最新编程资源的分享下载站】

运行测试

既然我们已经让DrinkMixer能够处理字典了，那么就继续行动，构建并运行这个应用吧。



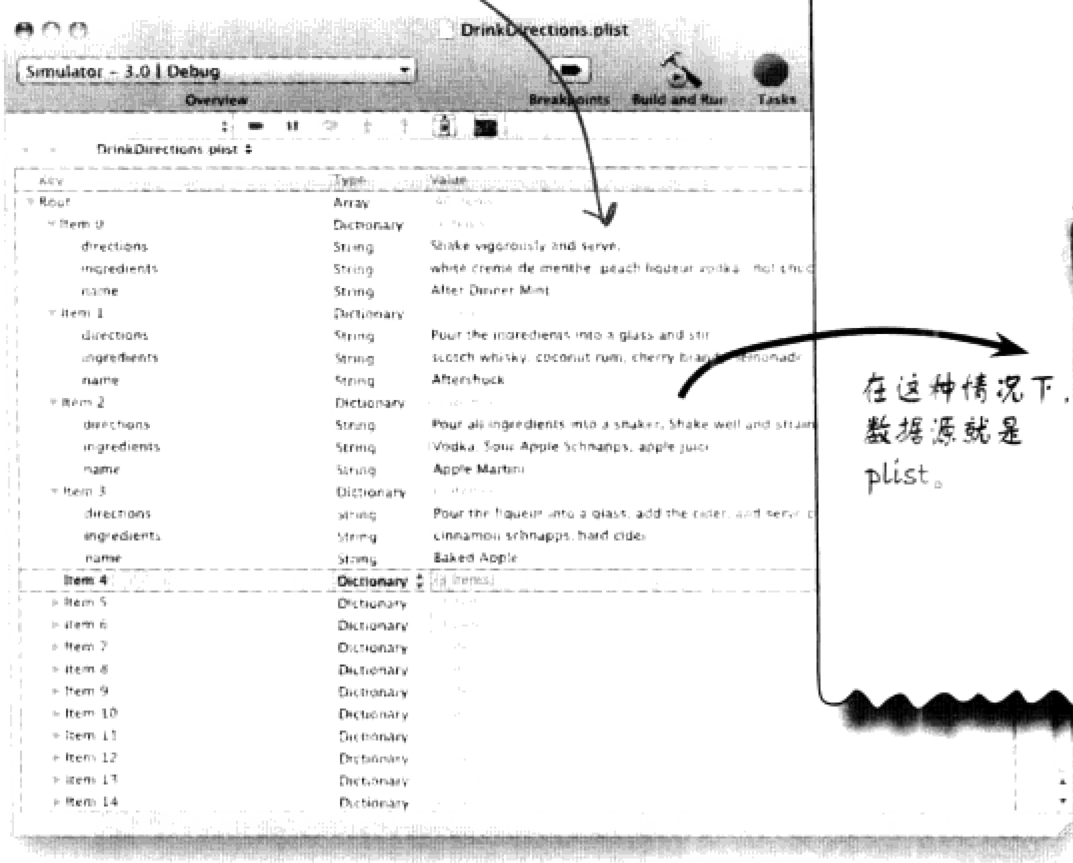


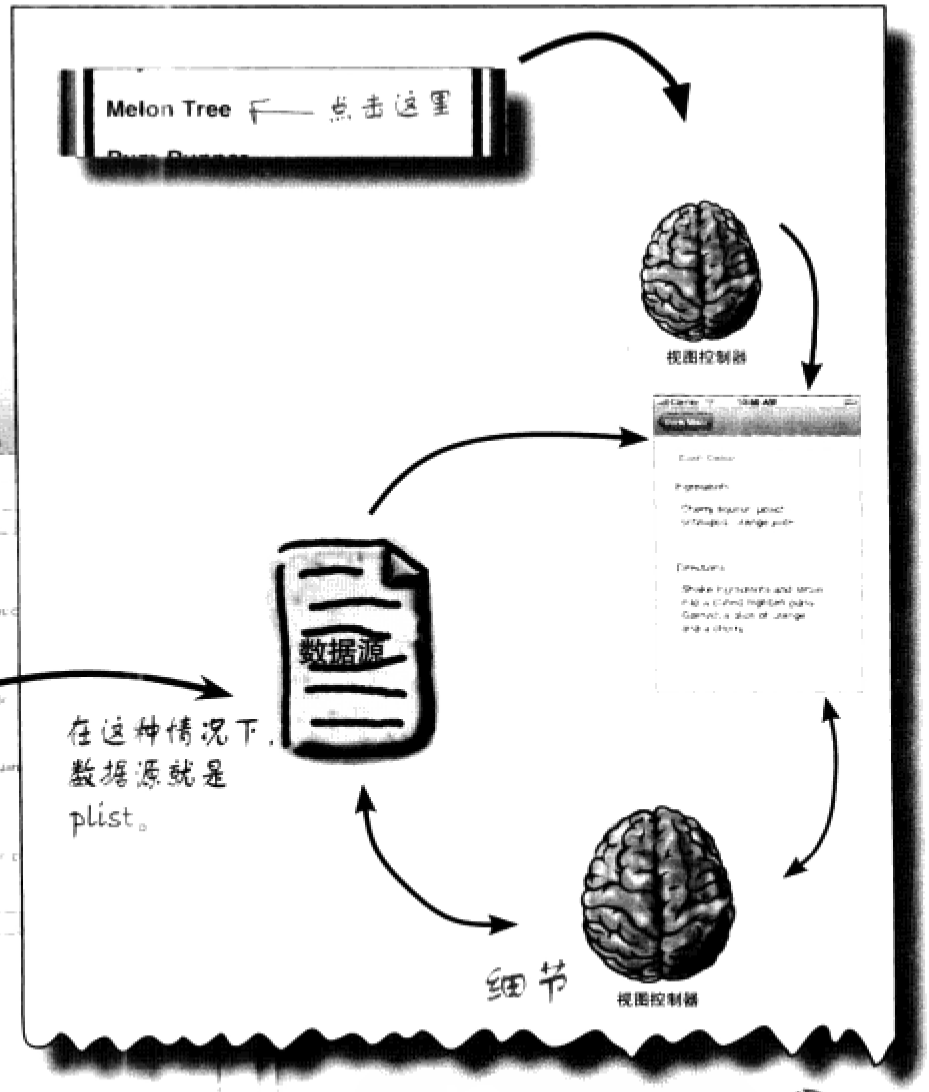
又可以正常工作了！已经不再崩溃，现在是时候填入细节了。

【拼吾爱-最新编程资源的分享下载站】

细节视图需要数据

现在你已经想办法解决了处理字典的问题，是时候把饮料的细节填进去了。但是要想把这些细节信息从字典数组中提取出来交给数据源，还需要另一个步骤。





脑力练习

我们怎样才能把信息从DrinkDirections.plist提取到应用中呢？

【拼吾爱-最新编程资源的分享下载站】

每个字典中都有我们需要的全套信息

目前为止我们仅仅是用name键将每种饮料的名字提取出来，再添加进应用中。为了将成份与调制说明也填充进去，我们需要使用其他的键。我们可以直接将这些键在代码中一写了事，但作为开发者我们应该要比这做得高明一些，所以要将其提取为常量。剩下的事就只有为细节视图控制器获得正确的字典了，然后它就可以提取出所需的信息了。开始行动，把这一切搞定！



练习

视图控制器需要对数据源进行直接访问，而要想得到这些数据，最简单的办法就是进行一些快速代码重构。

1 组织好你的字典常量，不要出错

由于我们在视图控制器中需要用到**name**、**ingredients**和**directions**等键，所以应该整理好代码，开始使用真正的常量。

创建一个名为**DrinkConstants.h**的新文件(在菜单中选择**File → New**，然后选择Other和空白文件)。为**name**、**ingredients**和**directions**添加常量定义(用#define)。在**DrinkDetailViewController.m**和**RootViewController.m**引入**DrinkConstants.h**。最后，将硬编码的@**"name"**替换为新的常量**NAME_KEY**。

2 设置细节视图控制器的drink属性

在单元格被点击时，你将对细节视图控制器进行实例化，之后需要在新的控制器中将drink属性设置为选中的饮料。

3 在细节视图控制器中添加代码，填充各字段

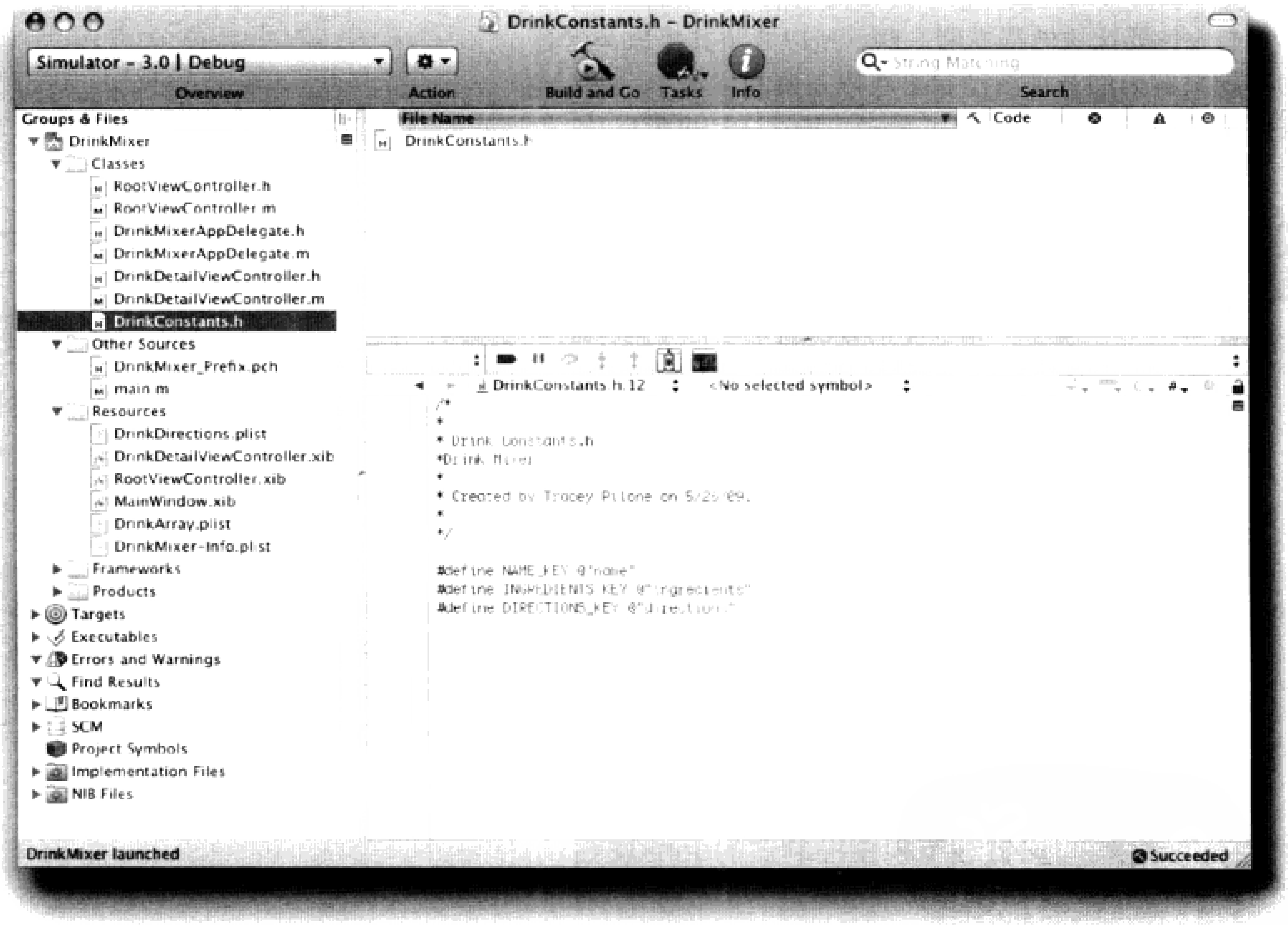
细节视图显示出来之前，视图控制器应该用饮料字典来设置各组件的内容，如名称、成份以及调制说明等。

【拼吾爱-最新编程资源的分享下载站】

练习答案

下面就是让细节视图正常工作所需要添加的全部代码。

1 DrinkConstants.h



DrinkDetailViewController.m和RootViewController.m都需要 `#import "DrinkConstants.h"`。

在这个文件中我们把字典键换成了常量...

Then add the constant to display the name:

```
    // Configure the cell.
    cell.textLabel.text = [[self.drinks objectAtIndex:indexPath.row]
objectForKey:NAME_KEY];
    return cell;
```

将@"name"换成了这个值。

RootViewController.m

【拼吾爱-最新编程资源的分享下载站】

❷ 设置细节视图控制器的drink属性

```
// Override to support row selection in the table view.
 - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSInd
exPath *)indexPath {

    // Navigation logic may go here -- for example, create and push
another view controller.
DrinkDetailViewController *drinkDetailViewController =
[[DrinkDetailViewController alloc] initWithNibName:@"DrinkDetailViewContro
ller" bundle:nil];
 drinkDetailViewController.drink = [self.drinks
objectAtIndex:indexPath.row];
 [self.navigationController pushViewController:drinkDetailViewController
animated:YES];
 [drinkDetailViewController release];
}
```

加入这行是为了从数组中取出一个字典。

RootViewController.m

❸ 在细节视图控制器中添加一个方法，用来填充各个域的值

```
 - (void) viewWillAppear: (BOOL)animated {
      [super viewWillAppear:animated];
      nameTextField.text = [drink objectForKey:NAME_KEY];
      ingredientsTextView.text = [drink
objectForKey:INGREDIENTS_KEY];
      directionsTextView.text = [drink objectForKey:DIRECTIONS_
KEY];
}
```

这个方法整个都是新的。

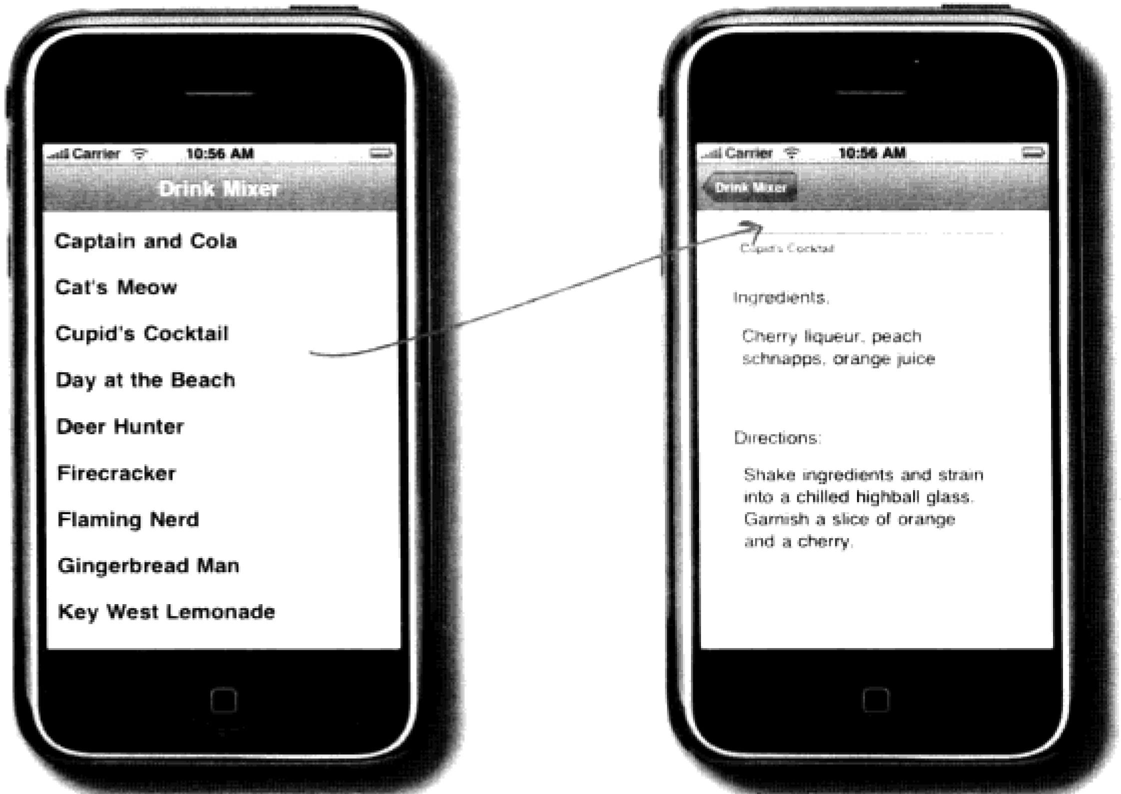
DrinkDetailViewController.m

【拼吾爱-最新编程资源的分享下载站】



运行测试

再次编译、构建并运行……



有问必答

问：每次有人点击了一种饮料时，我们都要重新创建细节视图。不能重用以前的视图吗？

答： 对于DrinkMixer来说，这不算什么大事；因为这个视图是很轻量的，我们在用户点击饮料时重建它也不会有太多开销。不过如果要追求性能的话，你可以重构这部分代码，每次都重用同一个细节视图控制器，只是将里面显示的内容换成被点击这一行的饮料。

问：我们为什么一定要把字典的键名都提取到一个单独的文件中呢？

答： 在代码里放一些硬编码字符串一般来说都不是什么好主意——不论你用的是什么编程语言、哪一种平台。把它们提取出来，用#define定义成常量，就可以让编译器对它们进行检查。这样当打错字的时候，把@"name"打成了@"nme"这样的错误只有在运行的时候才暴露出来，而把NAME_KEY写成了NME_KEY则会在编译的时候就会出错，从而提前发现错误。

问：我看了NSDictionary的文档，里面有一个valueForKey:方法，也有一个objectForKey:。二者有什么区别？

答： 问得好。valueForKey:是用在一种叫做键值编码的模式当中的，这种特殊的模式通常用于Cocoa Binding当中。这里有一个微妙的陷阱，那就是NSDictionary通常只是把对valueForKey:的调用简单地转化成一次objectForKey调用，而且看起来二者都能正常工作。然而，valueForKey 实际上会检查你传给它的键，根据键的不同会有不同的行为。这种情况几乎从来不会是你的意图所在（当然，除非你是在做一些Cocoa binding的活计）。正确的方法是使用objectForKey:。

【拼吾爱-最新编程资源的分享下载站】

Head First 休闲吧



看起来很有市场啊！
马上提交给苹果公司，然后……

【拼吾爱-最新编程资源的分享下载站】

自: iTunes商店

主题: DrinkMixer应用~~审查未通过~~

您的应用未被核准在App Store上发售。在您对表格视图的实现中，并未遵循苹果公司的人机接口指南。表格视图没有使用披露指示元素。未遵循人机接口指南的应用不能进行发售。请在修正您的实现之后，再重新提交您的应用以进行审查。

应该好好看看HIG了…

真的，如果你没遵守HIG的话，这种事可能并肯定会发生。嗯，笔者的一个朋友就遇到过…两次。



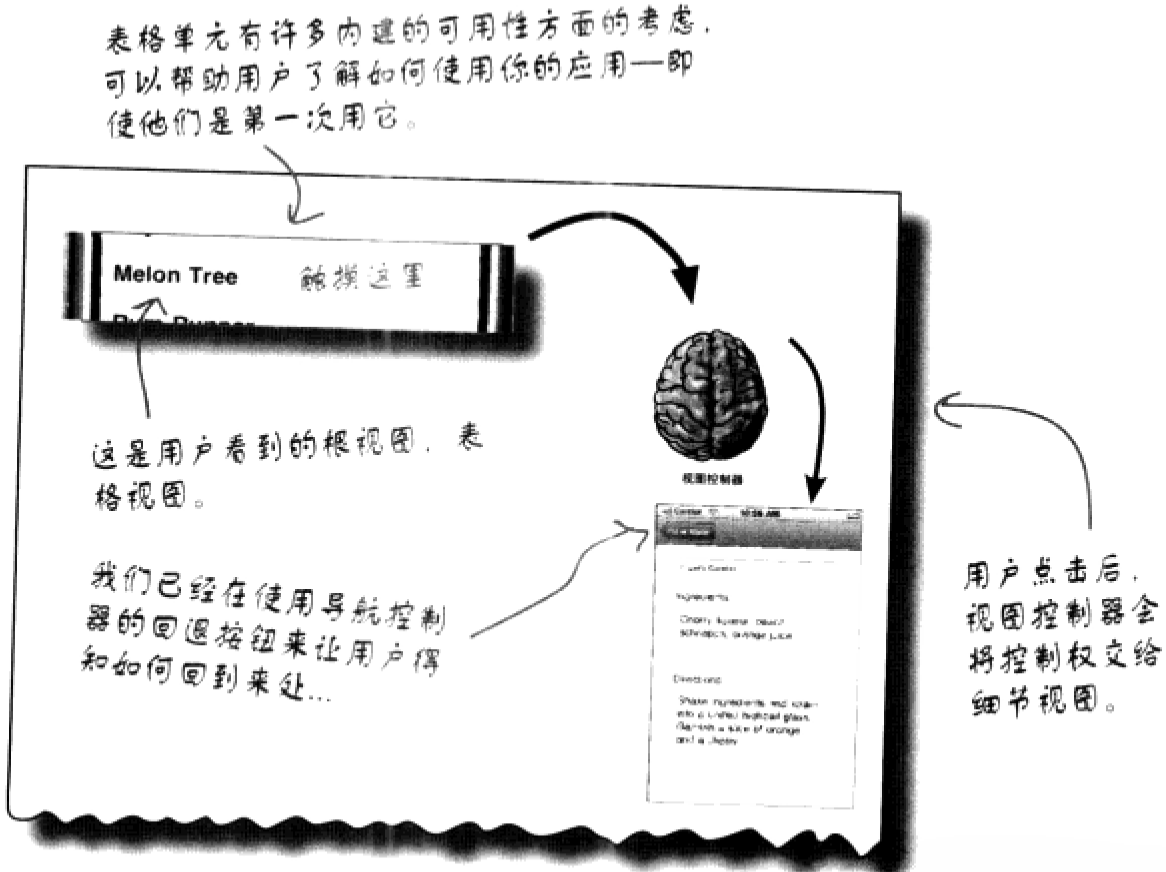
轻松一刻

稍后我们会从头到尾解释一遍审查流程。

稍后我们会在本书中带领你一步一个脚印地经历一遍提交应用进行审查的过程。目前，你只要操心怎么去修改DrinkMixer就行了！

我们在可用性上出了问题

我们知道用户需要触摸饮料的名字，才能单独看到每一种饮料的细节，但是用户怎么会知道这一点呢？对于如何处理逐层深入的数据，HIG有许多建议。我们使用表格视图就已经走上正确路线了，但是HIG还有许多额外的建议，去帮助用户理解如何在应用中遨游。



拿起笔来

是时候去深入了解HIG了，找出到底哪里出错了。

什么时候我们应当使用披露指示元素呢？

....

....

HIG提到了细节披露按钮和披露指示器——我们该用哪一个？为什么？

....

....

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

是时候去深入了解HIG了，找出到底哪里出错了。

什么时候我们应当使用披露指示元素呢？

在HIG第8章的“配置表格视图”这一节之中，会很快发现为什么说你违反了关于披露指示的规定：

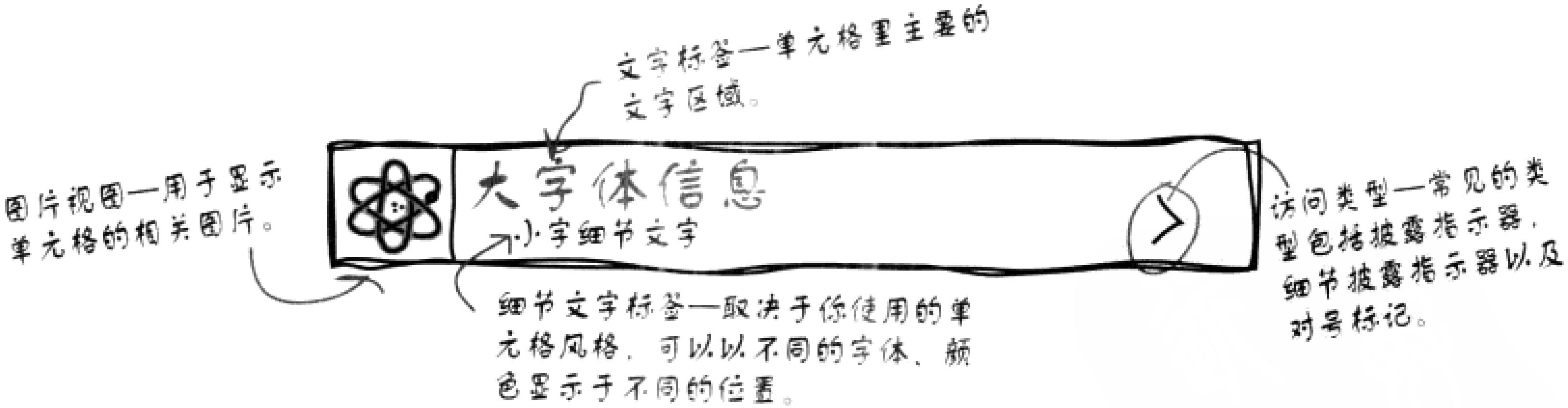
“披露指示元素……在你使用表格来表示分层数据时，是必需的。”

HIG提到了细节披露按钮和披露指示器——我们该用哪一个？为什么？

披露指示器表示对于一个条目还有下一层信息（比如饮料细节）可用，当你点击该条目时，就会选中这一行并显示出额外的数据。按钮则是在选中一行之外，还能做一些其他事情——比如触发某个动作。这已经超出了我们目前的需求，所以我们就坚持使用披露指示器了。

走近表格单元

那么，披露指示元素到底是什么东西呢？要把它放在哪里呢？还是让我们更仔细地再看一看HIG吧：



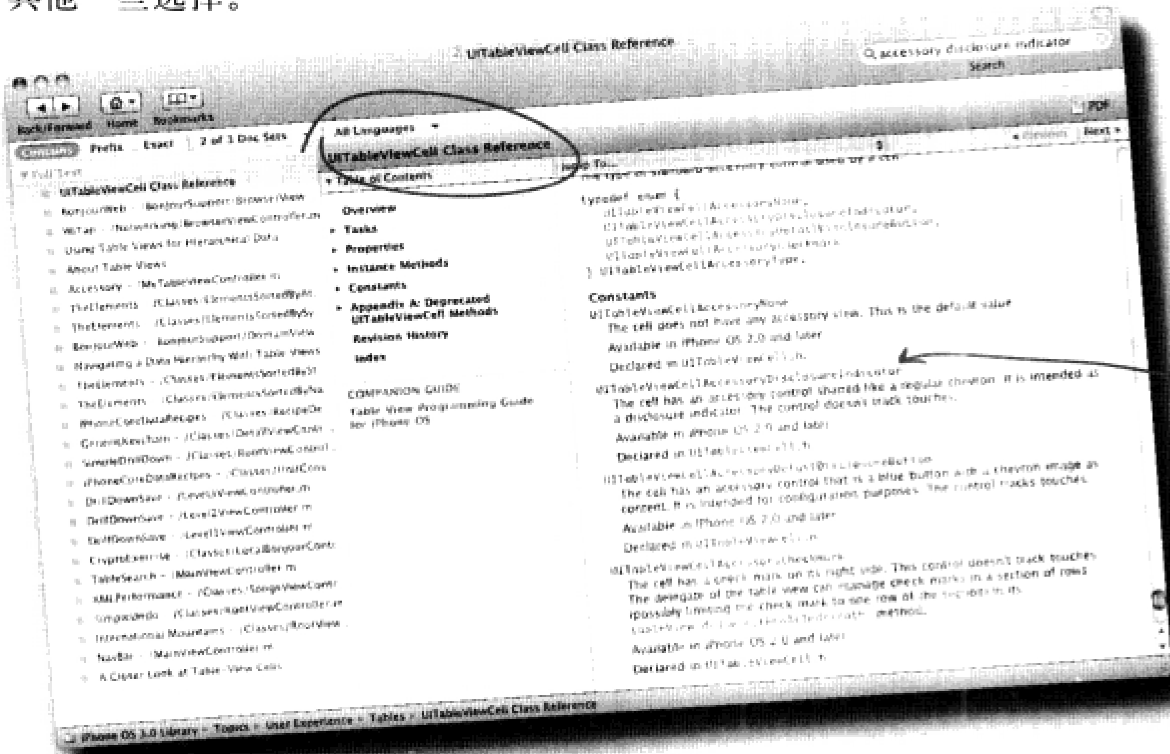
DrinkMixer使用的单元格非常简单，但是除了加上披露指示器之外，你还可以很容易地对单元格进行其他类型的定制，用于其他应用。甚至你可以令只支持一列的表格看起来好像有多列，比如加上一个缩略图等。如果有需要，你还可以调整字体的大小，为每个表格单元赢得一些空间。

非常精美的应用大多都对表格单元进行了某种类型的定制，在浏览API的同时请记住这一点。对于目前来说，我们只需要为单元格添加披露图标就可以了，这样就可以表示出用户点击它们的时候还有更多信息可看。

【拼吾爱-最新编程资源的分享下载站】

如果单元格会引导向更多信息，就用披露指示器

表格视图单元有许多内建的功能——我们现在只不过用到了一点皮毛。添加一个披露指示器只是小事一桩，只要告诉单元格要使用什么类型的附件图标就行了。看一看UITableViewCell的文档，里面讲到了其他一些选择。



这就是你要用的常量。

只要快速添加一行代码，在配置单元格的时候设置它的附件类型就可以了：

```
    //
Configure the cell.
    cell.textLabel.text = [[self.drinks objectAtIndex:indexPath.row]
valueForKey:NAME_KEY];
    cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
    return cell;
        }
```

只需将附件类型设置为对应的披露类型常量即可。

RootViewController.m



运行测试

继续行动，构建并运行……保证可以正常工作！

【拼吾爱-最新编程资源的分享下载站】

运行测试

只要一行简单的代码，就可以修正你遇到的App Store审查问题。



【拼吾爱-最新编程资源的分享下载站】



在重新提交到App Store之后，DrinkMixer出现在iTunes中！

销售报告 - DrinkMixer - 第1周

总销量 - 400次下载

单价 - $1.99

总收入 - $796

喔，仅仅一个星期！

记住，苹果公司还要从中抽头一定比例...

评论也接踵而至……

【拼吾爱-最新编程资源的分享下载站】

销售一度很强劲……

但是随后负面评论就开始出现了。这是怎么回事？

评价有点糟糕…



说的净是诸如“DrinkMixer太烂了－我没法加进任何东西”

……同时销量大跌！

另一条评论：“我需要的饮料比40个要多。”

“里面列出的饮料我一种都不喜欢。”

“我们酒吧有一些自制饮料，我不想再准备一份单独的饮料单了。”

“我打算换到iDrink了－虽然它更贵，但我能在里面添加新饮料，定制我自己的饮料单。”

【拼吾爱-最新编程资源的分享下载站】

拿起笔来

想一想当初你是怎样设计DrinkMixer的，再考虑这些反馈，设想一下接下来要做什么。

❶ 怎样才能解决用户关心的问题？

....

....

....

....

❷ 根据DrinkMixer的结构，你应该怎样对代码进行重构才能修正这些问题？

....

....

....

....

❸ 有没有更容易的修改方法？更麻烦的呢？

....

....

....

....

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

想一想当初你是怎样设计DrinkMixer的，再考虑这些反馈，设想一下接下来要做什么。

1 怎样才能解决用户关心的问题？

解决这些问题最简单的办法就是更新应用，让用户可以自己向饮料列表中添加更多内容。

2 根据DrinkMixer的结构，你应该怎样对代码进行重构才能修正这些问题？

我们可以加一个新的视图，让用户可以在里面输入他们自己的饮料信息。它看起来应该就像一个细节视图，但是可以让用户在里面打字输入想要的信息。我们必须能够将新信息保存起来，并更新表格来显示新饮料。

3 有没有更容易的修改方法？更麻烦的呢？

有许多种更麻烦的方法，以及很少几种又好又“容易”的方法。通常，对于我们而言添加这一功能最简单的方法就是尽量重用已经完成的代码。我们肯定能利用上导航控制器，现在来看看能不能用DetailDrinkView做一些有用的事……

脑力练习

你准备怎么实现这个视图，才能让用户可以向DrinkMixer中添加饮料？

【拼吾爱-最新编程资源的分享下载站】



搭建应用布局

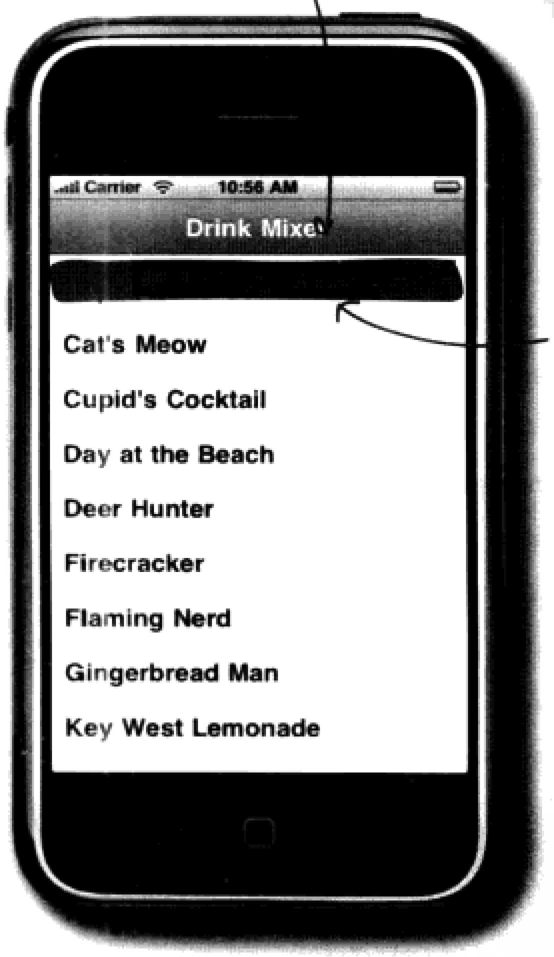
这里是为DrinkMixer的表格视图做出的两种设计。请问根据美观、易用和iPhone应用行为标准等原则，哪一种更好地向用户表示出了应该在哪里添加饮料？

在导航控制器上加入一个某种类型的按钮来启动一个新视图。

在导航控制器下面添加一个新的工具栏，上面带有一些按钮。



选择1



选择2

你有足够的空间放下多个按钮，除了添加饮料按钮，你还可以在需要时加入其他按钮。

哪个界面更好？ ....

为什么？（详细点） ....

....

为什么不用另一个？ ....

【拼吾爱-最新编程资源的分享下载站】



搭建应用布局答案

这里是为DrinkMixer的表格视图做出的两种设计。请问根据美观、易用和iPhone应用行为标准等原则，哪一种更好地向用户表示出了应该在哪里添加饮料？



选择1



选择2

哪个界面更好 选择1。

为什么？（详细点）因为在导航控制器上放一个图标并不会从表格视图中占去太多空间。而且导航控制器中也内置了对这个按钮的支持。

为什么不用另一个？选择2令界面有些局促，而且也要写更多代码。

【拼吾爱-最新编程资源的分享下载站】

用导航控制器按钮进行编辑

到目前为止，导航控制器已经被用来在视图之间进行切换。但是如果你用过一段时间其他的iPhone应用，就会知道它的功能远不止于此。由于UITableView几乎总是嵌在一个导航控制器中的，对表格进行编辑通常都是通过控制器上面的按钮来完成的。首先，让我们在导航控制器上面添加一个加号（+）按钮，用户点击这个按钮就可以添加饮料。



用户可以点击这个加号按钮来添加饮料。

拿起笔来

用Xcode在导航控制器上添加按钮，以及与之关联的IBAction和IBOutlet。

1 在Interface Builder中打开RootViewController.xib。

在控件库中找到Bar Button Item，并拖曳一个按钮到主窗口中（这会在列表中把它加到表格视图的后面）。这个按钮并不会显示在Interface Builder中的导航控制器上——我们得添加代码，让它在运行时显示出来。

不显示是因为Interface Builder里的导航控制器是模拟的，而不是真的。

2 为addButtonItem添加IBAction、IBOutlet和属性声明。

像其他按钮一样，我们得有一个IBAction让按钮被点击的时候调用，还得有一个对按钮自身的引用——全都在RootViewController.h里。

3 为addButtonItem添加synthesize、dealloc以及addButtonPressed方法。

合成属性，释放引用，并实现一个addButtonPressed方法，在按钮被点击时记录日志信息——这些都在RootViewController.m里。

4 在Interface Builder中做收尾工作。

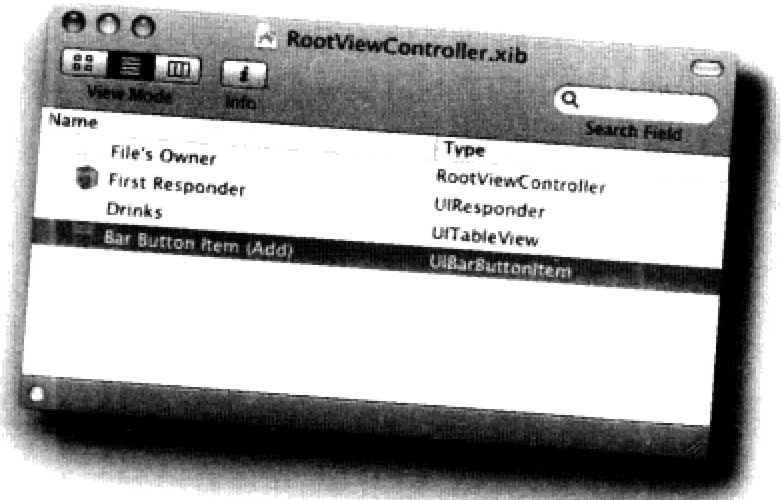
再次打开RootViewController.xib，将新加入的导航栏按钮与主窗口中的动作和输出口关联起来。

最后，为导航栏按钮调出查看器，将**Identifier**这一项改成**Add**。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

用Xcode在导航控制器上添加这个按钮，以及与之关联的IBAction和IBOutlet。



1. **在Interface Builder中打开RootViewController.xib。**
   在控件库中找到Bar Button Item，并拖曳到主窗口中（按钮会被添加到列表中）。

2. **为addButtonItem添加IBAction、IBOutlet和属性声明。**

```
@interface RootViewController : UITableViewController {
    NSMutableArray *drinks;
    IBOutlet UIBarButtonItem *addButtonItem;
}
@property (nonatomic, retain) NSArray *drinks;
@property (nonatomic, retain) UIBarButtonItem *addButtonItem;
- (IBAction) addButtonPressed: (id) sender;
@end
```

RootViewController.h

3. **为addButtonItem添加synthesize、dealloc以及addButtonPressed方法。**

```
@synthesize drinks, addButtonItem;
```

RootViewController.m

```
- (IBAction) addButtonPressed: (id)sender {
    NSLog(@"Add button pressed!");
}
```

【拼吾爱-最新编程资源的分享下载站】


```
self.
navigationItem.rightBarButtonItem = self.addButtonItem;
```

这一行放在viewDidLoad里面。

```
- (void)dealloc {
        [drinks release];
        [addButtonItem release];
    [super dealloc];
  }
```



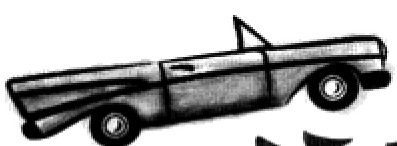
RootViewController.m

4 在Interface Builder中做收尾工作。

再次打开RootViewController.xib，将新加入的导航栏按钮与主窗口中的动作与输出口关联起来，右键单击后使用弹出的菜单。

最后，为导航栏按钮调出查看器，将**Identifier**这一项改成**Add**。





运行测试

继续行动：构建并运行应用……

【拼吾爱-最新编程资源的分享下载站】

运行测试

继续行动：构建并运行应用……



按钮起作用了！现在你可以在控制台中看到一条确认信息…

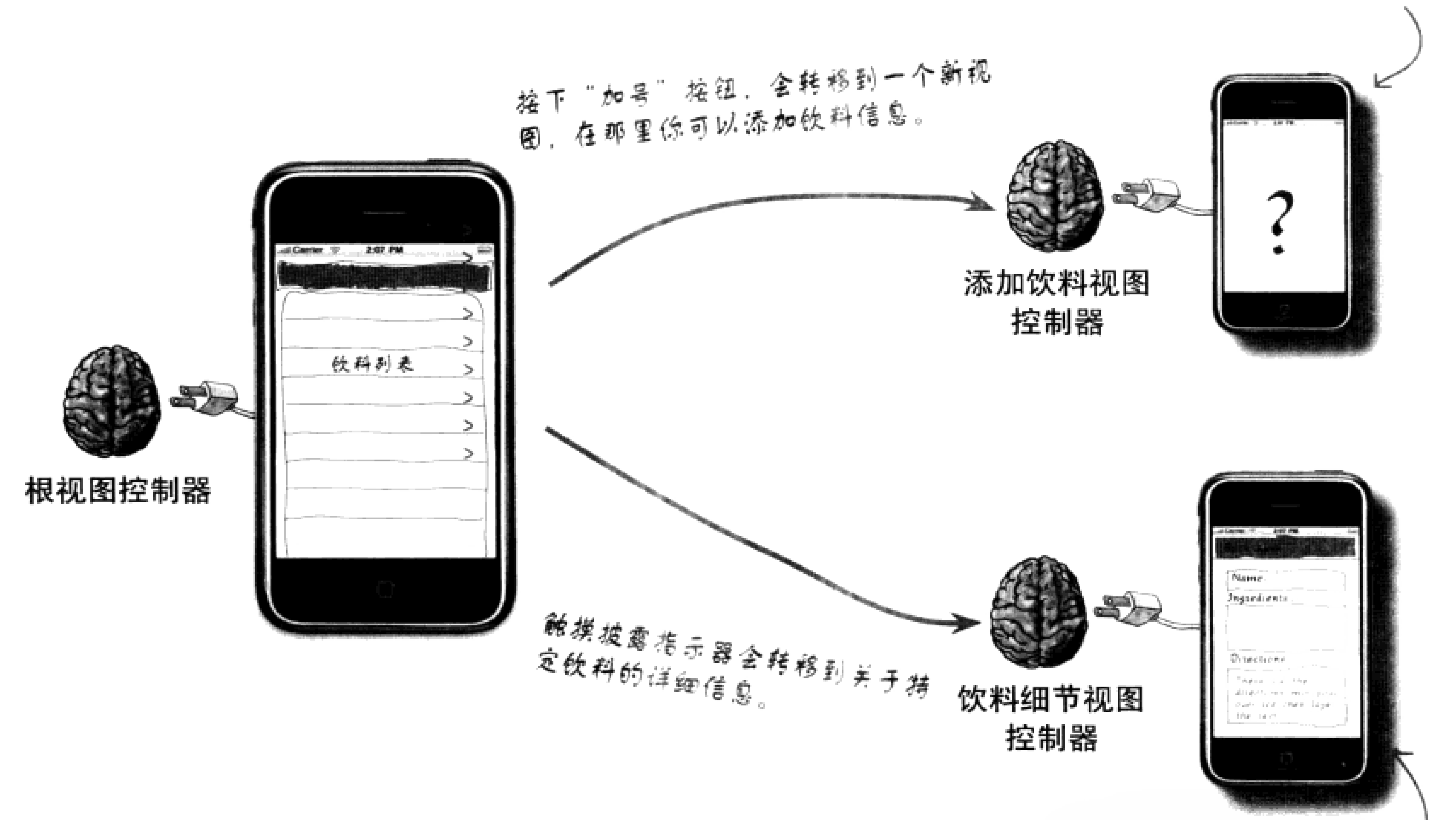


按钮出现在了视图中，但是然后该怎么办？

【拼吾爱-最新编程资源的分享下载站】

这个按钮应当创建一个新视图

我们的新按钮起作用了：动作方法被调用了，但是什么有用的事都还没做。我们需要给用户一个可以输入新饮料信息的地方，这可以在一个新视图中来完成。正如细节视图一样，我们可以让导航控制器来处理视图的转换。



脑力举重

要让用户可以输入新饮料，我们需要哪些东西？确切地说，需要哪些控件，而你又要如何布置它们？视图控制器应该如何运转？

....

....

....

....

【拼吾爱-最新编程资源的分享下载站】

我们需要一个视图……但并不一定是一个新的视图

我们的“新饮料”视图实际上只不过是细节视图的一个可编辑版本。所以我们并不会从头创建一个新的nib，而是利用UI(nib文件)是与其行为(.m文件中UIView子类)分离的这种情况，来重用细节视图。

到目前为止，我们一直都在nib和视图控制器之间维持着一对一的关系。这非常正常，可是视图控制器实际上都是正常的Objective-C类，我们可以用面向对象的扩展机制——比如继承——来为之加入我们期望的行为。



添加饮料视图控制器

DrinkDetailViewController.xib



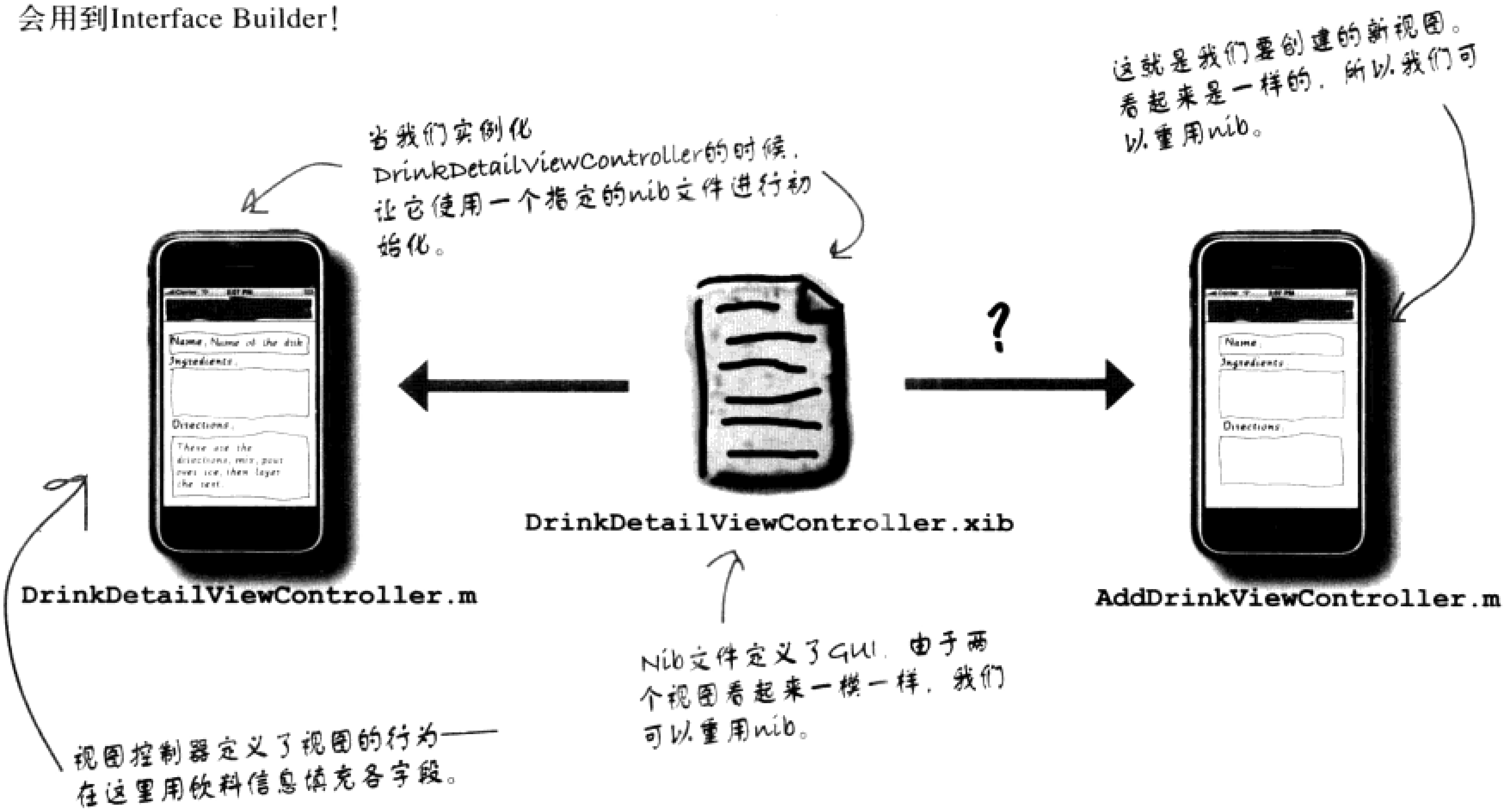
并不一定。

要记住，nib文件只不过是一个用XML方式来表示的视图。使用nib要比用代码来布置视图简单得多。而由于nib中只是图形信息，你需要把实际代码放在其他地方。也就是视图控制器的所在……

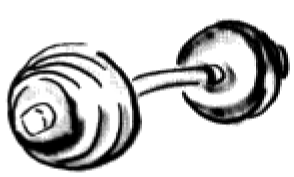
【拼吾爱-最新编程资源的分享下载站】

视图控制器定义了视图的行为

从用户的角度来说，我们有3个视图：表格视图、细节视图以及新饮料视图。但是由于我们重用了.xib文件来创建“新”视图，所以只需要添加一个新的视图控制器就可以支持添加饮料了。也就是说根本不会用到Interface Builder！



将UI与行为分离，有助于对视图进行重用。



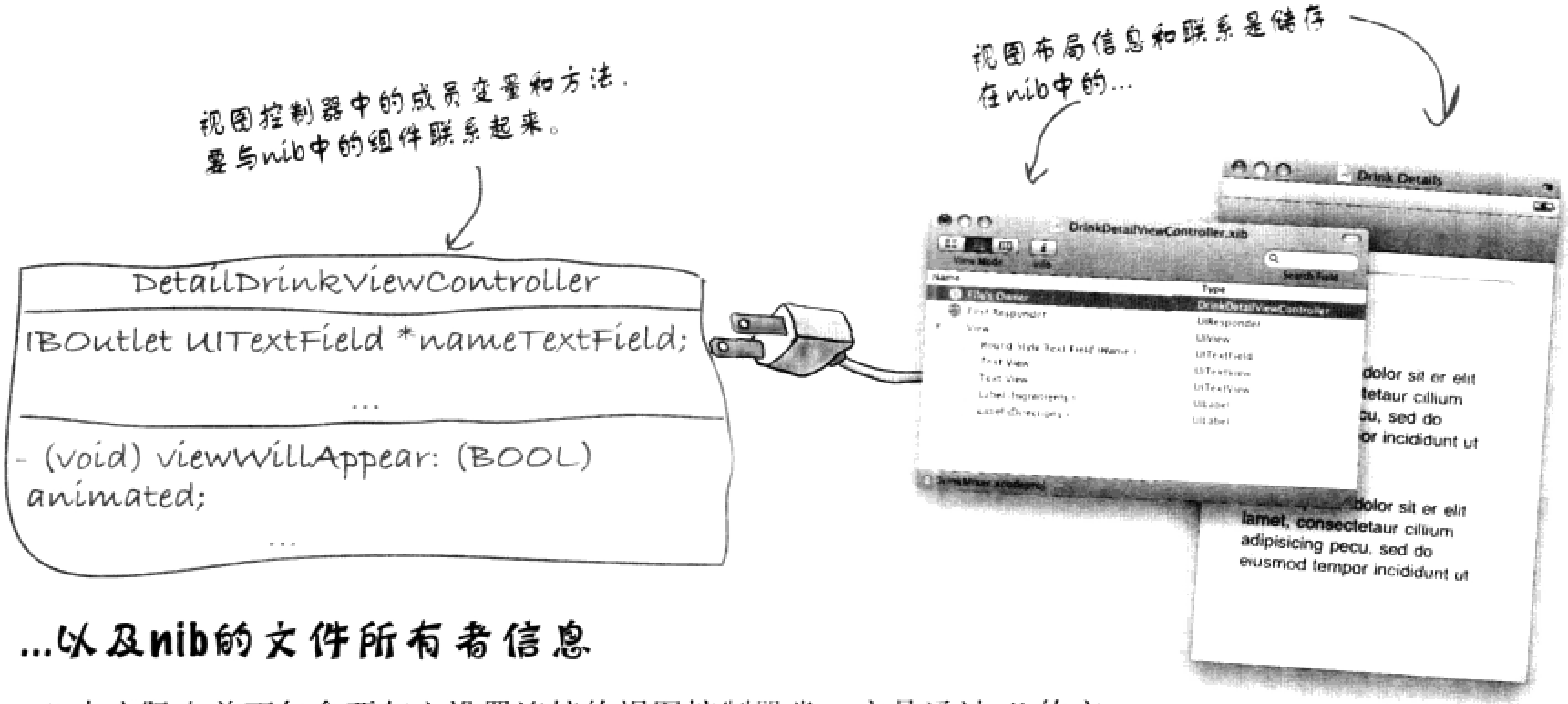
脑力举重

同时重用nib文件与细节视图控制器也是一种选择……
但是在什么地方我们会碰到问题呢？

【拼吾爱-最新编程资源的分享下载站】

Nib文件包含着UI组件与连接……

重用nib的方式之一是创建一个新的视图控制器，并在它进行初始化的时候，将DrinkDetailViewController.xib文件传给它。不过这样做有几处难点。记得吗，我们用Interface Builder并不是仅仅进行界面布局的；我们还用它建立起组件与加载nib的类之间的联系。



...以及nib的文件所有者信息

nib中实际上并不包含要与之设置连接的视图控制器类。它是通过nib的文件所有者来获取这一信息的。当你将nib传递给视图控制器时，控制器会将nib反序列化，然后开始按照nib文件中存储的输出口的名字建立联系。也就是说如果我们把nib传给另一个视图控制器，我们需要保证相同的输出口的名字都一样，对应的动作也一样等。



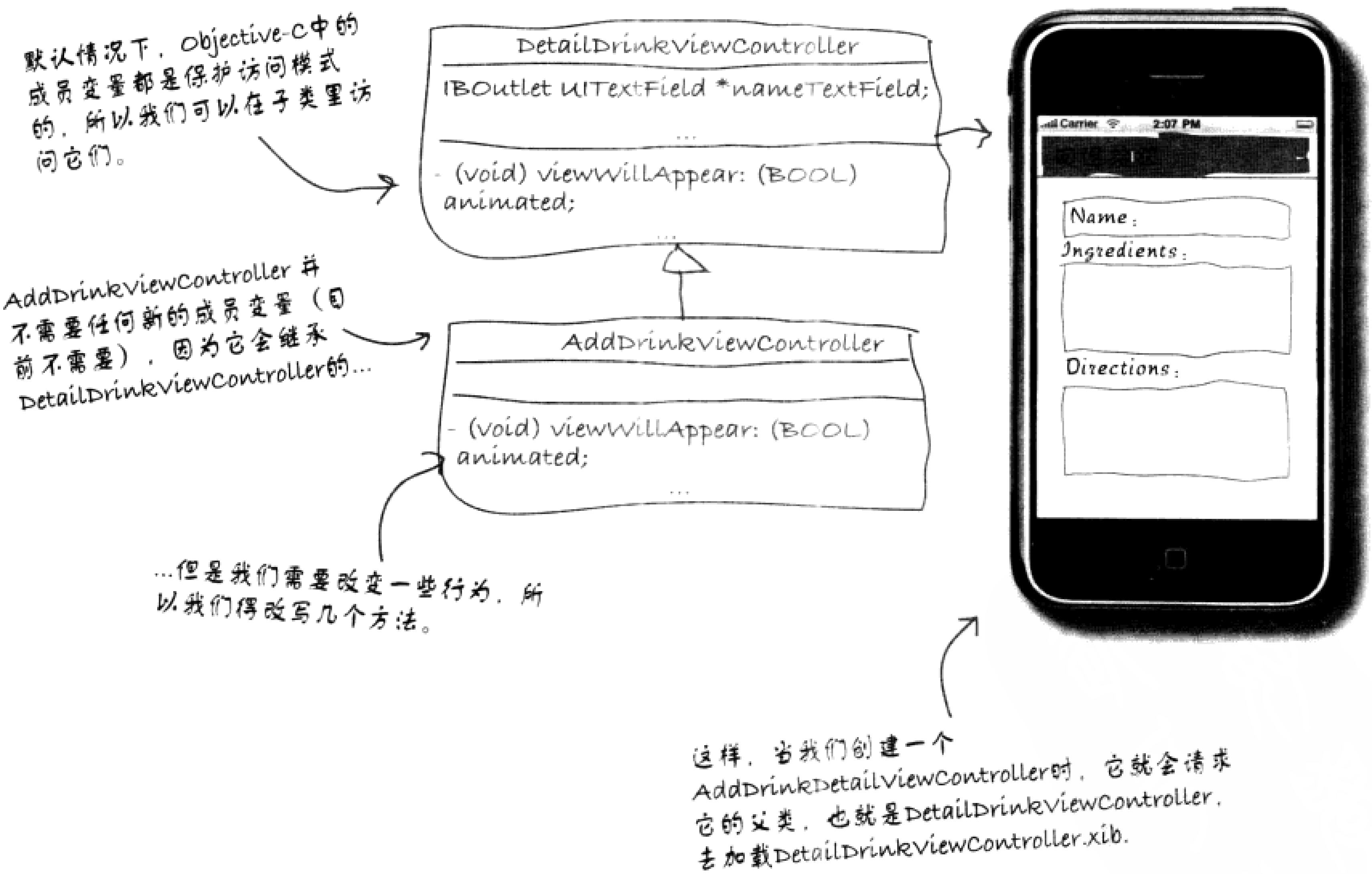
重用nib可以满足我们在这个应用的需求，但是这并不适用于所有应用…

受DrinkMixer的构建方式的影响，我们只要对细节视图做派生就可以满足需要。这种方法对于这个应用非常好用，但是在更复杂的应用中这样做就要小心，因为代码可能会难以维护。通常，最好还是吃点苦头，打造一个新视图……有些时候你会意识到这些视图并不应该是一模一样的。

【拼吾爱-最新编程资源的分享下载站】

你可以对视图进行派生和扩展，就像对其他类一样

我们并不是仅仅重用nib，同时将所有输出口与动作方法都重新创建一次，而是派生出DetailedViewController的子类，并加入我们需要的行为。我们的AddDrinkViewController与DetailedViewController相同，只不过具有了创建与保存一个完整的新饮料的能力。其他一切行为——显示名称、显示描述等——都是与DetailedViewController完全一样的。



首先，我们需要创建新的视图控制器。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：我还是弄不明白，新的视图控制器怎么能没有一个新nib。

答： 在那个nib当中，没有什么东西是不能通过一般的Objective-C代码手工创建的。你很可能已经发现，用Interface Builder通过nib来对视图进行布置，要比用代码进行轻松得多，所以当你要创建一个新视图的时候，通常都会通过创建一个nib来完成。但是实际上你完全可以构建一个应用程序，却一个nib也不用。对于我们来说，准备在二者之间取中庸之道：我们会创建一个新视图，但是重用另一个视图的UI信息。

问：那为什么“留神”一节警告说重用nib要小心？这样做到底好不好？

答： 很不幸，答案是：看情况。对于DrinkMixer，我们可以重用DetailDrinkView和它的nib，是由于我们期望两个视图的布局完全一致，而DetailDrinkView实际上也没有任何特殊功能。然而在一个更复杂的应用程序中，你这样做就可能会遇到问题，比如不停地折腾这两个视图控制器，或者向子类暴露了太多信息以致于代码变得难以维护。这并不是iPhone开发所独有的问题；不管怎样，你要派生一个子类的时候都必须要小心从事。对于我们的应用，使用子类很合适，这种用法你在苹果公司的一些例子应用程序中也会见到（这也是为什么我们会在本书中包含这些内容的部分原因）。但是同样有可能在其他一些应用程序中，你会希望视图是相似的，但又不是完全相同。在这些情况下，请同时创建新的视图控制器与nib。

【拼吾爱-最新编程资源的分享下载站】

用Xcode在没有nib的情况下创建视图控制器

下面我们要做的，就是在Xcode中创建一个新的、没有自己的nib的视图控制器，然后把它改为继承自DetailDrinkViewController。这个新视图会继承所有的成员变量、行为（这个我们要有改动），以及我们所需的nib。

拿起笔来

来到Xcode中，创建 AddDrinkViewController的各个文件。

- [ ] 在菜单中选择 New → File... ，创建一个新的、没有对应nib的UIViewController子类，命名为AddDrinkViewController。

注意创建新文件时的选项...你不需要随视图创建xib。

- [ ] 打开新建的AddDrinkViewController.h文件并进行修改，将继承的基类从UIViewController改为DetailedDrinkViewController。别忘了引入DetailedDrinkViewController.h文件。

我们马上就会添加所需的新行为...

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

来到Xcode中，创建 AddDrinkViewController的各个文件。

你需要在菜单中选择File → New之后弹出的对话框中，创建一个新的UIViewController子类。请一定不要选中With XIB for user interface点选框，因为我们不需要这个.xib文件。

为了使用DetailDrinkViewController，需要引入它的头文件，这样编译器才能明白我们是在说什么。

```
#import <UIKit/UIKit.h>
#import "DrinkDetailViewController.h"

@interface AddDrinkViewController : DrinkDetailViewController{

}
@end
```

默认我们的视图控制器是继承自UIViewController的。改成DrinkDetailViewController。

AddDrinkViewController.h

AddDetailViewController.m文件可以就按照Xcode生成的原样不变。

有问必答

问： 等一下，为什么我们不直接把nib传递给AddDrinkViewController呢？为什么要派生子类呢？

答： 我们也可以那样做，但是问题是我们并不仅仅在处理GUI布局。我们还有文本字段和标签有待填充内容。DetailedDrinkViewController已经拥有所有我们需要的字段对应的输出口，而且它还有在显示饮料前用饮料的信息填充字段的功能。如果我们不使用子类，就必须在新视图控制器中重新实现这些功能。

问： 这是不是某种精心设计的Head First例子啊？还是说我真的应该给予关注？

答： 你真的应该留心这个问题。这种模式出现得还是颇为频繁的，许多苹果公司的示例应用程序都用到了它。用一个视图来显示数据而用另一个进行编辑，是很普遍的，特别是在表格驱动的应用程序当中，用户可以将表格置于编辑模式中（我们稍后会进一步谈论这个问题）。有时候你应该使用完全不同的视图；有时你可以重用现有的。

问： 你提到了默认情况下成员变量是保护访问模式的。如果我想在类里使用私有成员该怎么做？

答： 很简单——只要在接口定义中成员变量声明的前面加上@private（或者是@public，如果需要公有成员的话）。如果你没有在前面加上一个访问限定符，Objective-C默认将成员设为保护模式的。

【拼吾爱-最新编程资源的分享下载站】



吉姆：现在我们有AddDrinkViewController这个类了，那么我们要做的就是把它压入视图栈中了，就像对细节视图那样，是吧？

乔：有道理——我们是用导航控制器将细节视图压入栈中来进入细节数据的……

弗兰克：不过再把一个新视图放到我们的列表里面，就有点不一样了。

吉姆：为什么？

弗兰克：嗯，添加新的饮料实际上是一个分支任务。

乔：啊？

弗兰克：用户离开了浏览饮料这个常见工作流，来创建一种新饮料。

乔：哦，对啊。现在他们是在输入饮料了，而不是阅读和调制饮料。

弗兰克：对，所以这种时候，重要的是通知用户，他们必须完成这个任务。要么完成所有步骤，要么——

乔：——要么取消任务。

弗兰克：那么，这个视图应该是什么样的呢？

脑力练习

下面这些视图中，哪一种更适合通知用户该做什么？哪一个与其他相比，更容易引起误解？



【拼吾爱-最新编程资源的分享下载站】

模式视图让用户把注意力集中在手头的任务上……

当用户把玩你的应用时，他们都习惯于在浏览数据时见到视图来回进出。然而，某些任务是不同于一般的数据浏览的，对于这些任务我们必须要唤起用户的关注。在iPhone上，可以通过模式视图来做到这一点。这些视图从开发者的角度来说，仍然是正常的视图，但是给用户的感受却有以下这些不同之处：



...比如添加或者编辑条目

我们打算在用户要向DrinkMixer中添加新饮料的时候，使用模式视图。他们必须保存添加的饮料，或者放弃（取消）修改，然后才能返回到DrinkMixer主应用。

【拼吾爱-最新编程资源的分享下载站】

任何视图都可以作为模式视图呈现

到目前为止，我们都是在使用导航控制器来显示新视图。对于模式视图来说略有不同：任何UIViewController都可以作为模式视图来显示，然后在需要时再隐藏起来。要在当前视图上面显示一个模式视图很简单，只要向当前视图发送一则presentModalViewController:animated:消息即可。由于我们的RootViewController就是需要显示模式视图的视图控制器，所以可以向**self**发送这个消息，如下所示：

```
[self presentModalViewController:addViewController animated:YES];
```

self是Objective-C中的关键字，指代当前正在执行方法的对象。这与Java和C++中的this类似。

这就是你要作为模式视图来显示的视图控制器，对于我们来说，就是新建的AddDrinkViewController。

如果你用NO作为animated的实参，那么视图只会简单地显示出来。如果用YES，我们就会得到从底部平滑进入的效果。

拿起笔来

修改RootViewController.m文件，让AddDrinkViewController能够在加号按钮被点击时，显示在UINavigationController中。

- [ ] 你需要引入AddDrinkViewController.h，这样RootViewController才能了解这是一个什么样的类。

- [ ] 修改addButtonPressed:sender:方法，创建一个 AddDrinkViewController，并将其作为模式视图呈现出来。要留意内存的管理问题——不要忘记释放控制器的引用，造成泄露。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

修改RootViewController.m文件，让AddDrinkViewController能够在加号按钮被点击时，显示在UINavigationController中。

```
#import "RootViewController.h"
#import "DrinkConstants.h"
#import "DrinkDetailViewController.h"
#import "AddDrinkViewController.h"
```

这儿没啥可说的—就是引入个文件。

RootViewController.m

```
- (IBAction) addButtonPressed: (id) sender {
    NSLog(@"Add button pressed!");

    AddDrinkViewController *addDrinkVC = [[AddDrinkViewController alloc] in
itWithNibName:@"DrinkDetailViewController" bundle:nil];
    [self presentModalViewController:addDrinkVC animated:YES];
    [addDrinkVC release];
}
```

分配AddDrinkViewController和分配DetailedDrinkViewController没什么两样——记住，这是个子类。它也会使用同一个nib。

现在我们要做的就是显示模式视图—由于RootViewController也是一个视图控制器，所以只要调用presentModalViewController就行了，iPhone会处理其余的工作。

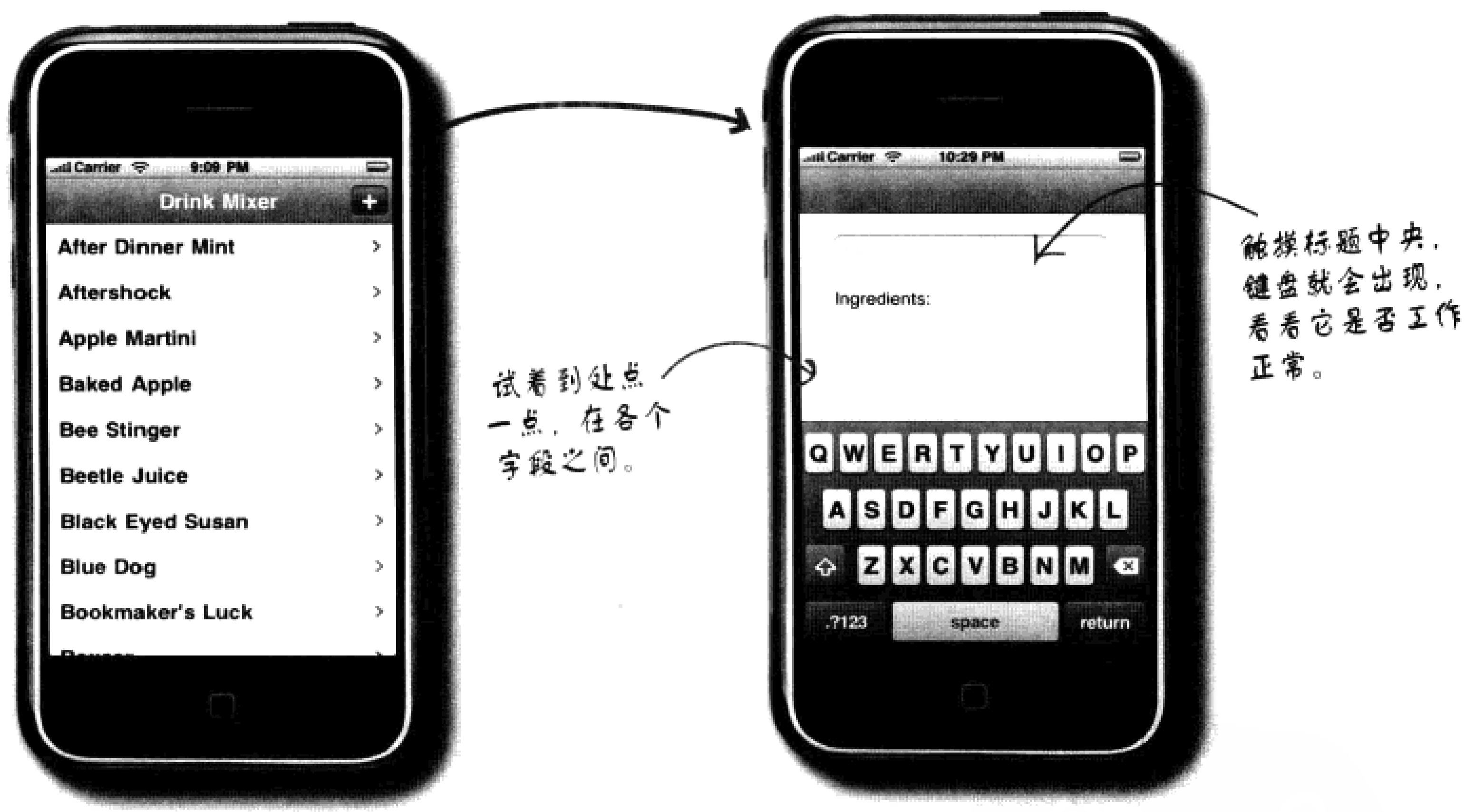
在显示新视图控制器时，RootViewController会持有一个对它的引用。不要忘记释放这个引用！

RootViewController.m

【拼吾爱-最新编程资源的分享下载站】

运行测试

现在添加饮料视图已经完整实现了，构建并运行这个项目。请确保测试了所有功能：滚动列表、深入细节以及最后的添加饮料。一定要试着添加一种新的饮料名称……



如果你的键盘不能用，可能是因为字段仍然处于不可编辑的状态。

留神！

之前在第4章中，我们让你在Interface Builder里把这些字段设成了不可编辑。如果键盘不出现，请试着回到Interface Builder中，检查一下现在这些字段是不是可以编辑的。

但是当你输入完之后，要怎么办呢？

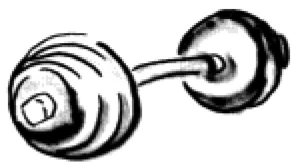
【拼吾爱-最新编程资源的分享下载站】



这是个问题。

实际上，这是两个互相关联的问题。添加饮料细节视图需要能够以两种方式消失：要么用户取消了输入，要么保存了饮料。我们得处理这两种情况。

【拼吾爱-最新编程资源的分享下载站】



脑力举重

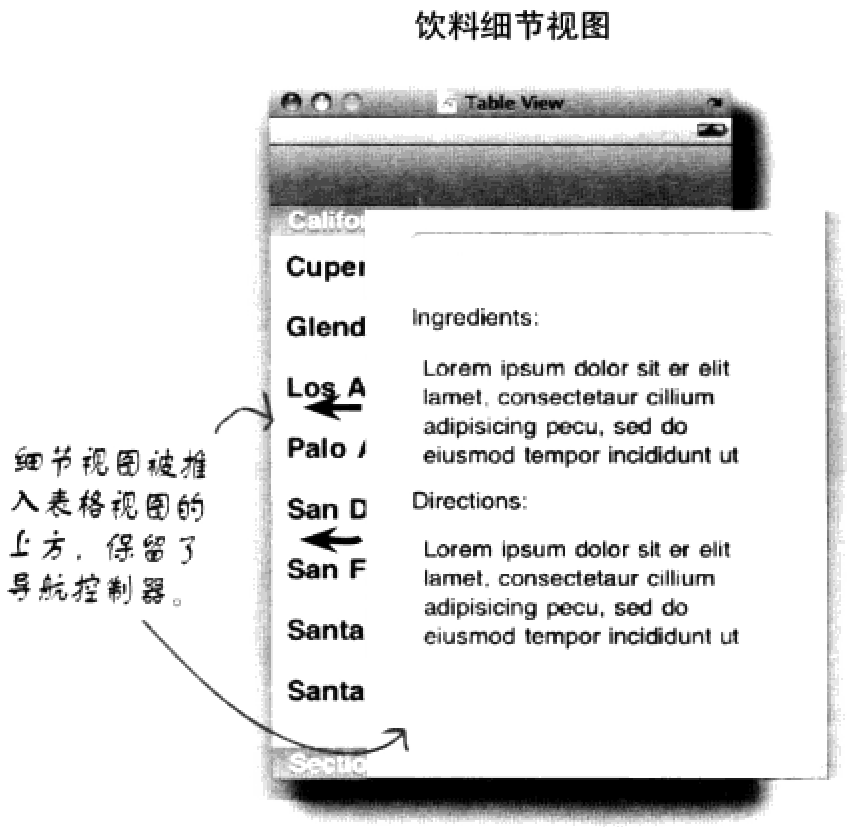
我们应该把保存和取消按钮放在哪儿呢？

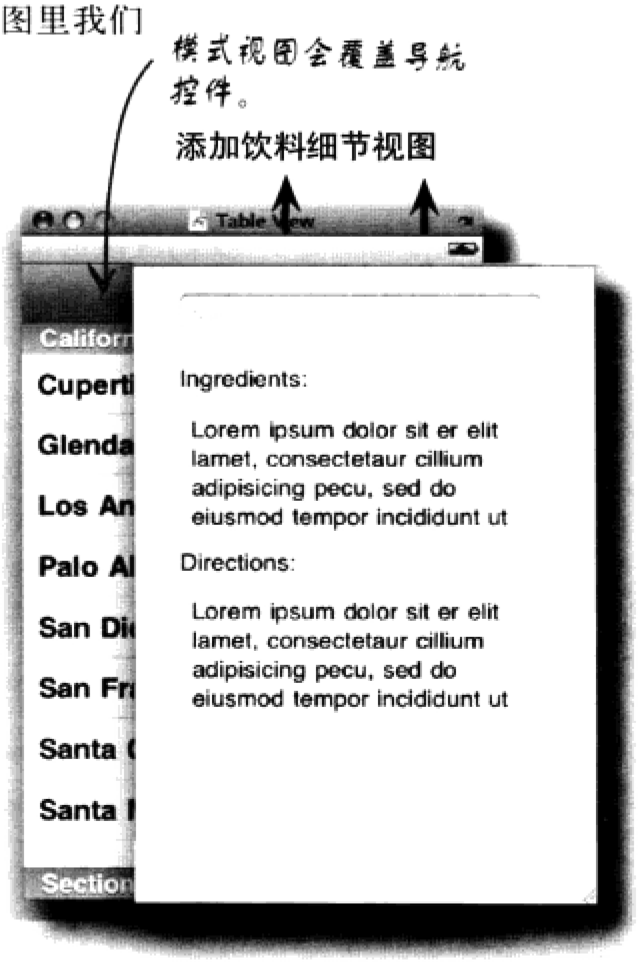


【拼吾爱-最新编程资源的分享下载站】

我们的视图没有导航栏

为了与DrinkMixer的其他部分保持一致，我们确实应该把保存和取消按钮放在视图顶部的导航栏当中。问题是，在模式视图版本的细节视图里我们没有导航栏。





我们可以自己手工添加一个控制栏，但是要记住我们使用的是共享的饮料细节视图的nib，里面的导航栏是来自导航控制器的。由于我们将添加饮料视图显示为模式视图，导致导航栏被覆盖。

不要试图去从饮料细节视图内部来解决这个问题，相反地，我们可以为我们的添加饮料视图内嵌一个自己的导航控制器，如下所示：

我们不去显示自己的addDrinkVC，相反却显示一个addNavCon视图控制器。

这样就可以用一个导航控制器将添加饮料视图包装起来。

```
UINavigationController *addNavCon = [[UINavigationController alloc]
initWithRootViewController:addDrinkVC];
```

【拼吾爱-最新编程资源的分享下载站】

按图索骥！

分配AddDrinkViewController就和分配DetailedDrinkViewController一样——记住，这是一个子类。它也使用同一个nib。

分配UINavigationController，将AddDrinkViewController传递给它作为根视图控制器。它会持有该控制器，因为需要将其显示出来。

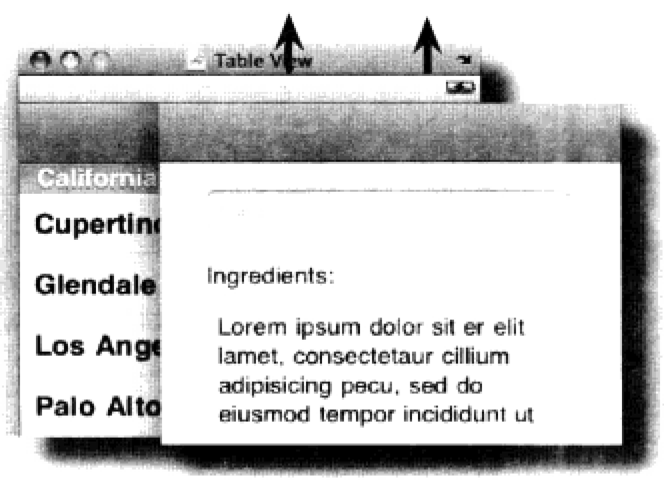
```
- (IBAction) addButtonPressed: (id) sender {
    NSLog(@"Add button pressed!");

    AddDrinkViewController *addDrinkVC = [[AddDrinkViewController alloc]
initWithNibName: @"DrinkDetailViewController" bundle:nil];
    UINavigationController *addNavCon = [[UINavigationController alloc] ini
tWithRootViewController:addDrinkVC];
    [self presentModalViewController:addNavCon animated:YES];
    [addDrinkVC release];
    [addNavCon release];
}
```

现在我们只要显示模式视图即可——RootViewController也是一个视图控制器，所以只要调用presentModalViewController就行了，iPhone会处理其余的工作。

不要忘记释放AddDrinkViewController和NavigationController的引用。

RootViewController.m



现在成了！模式视图也有了导航控制器，有地方放你的按钮了。现在我们只要创建这些按钮就行了……

【拼吾爱-最新编程资源的分享下载站】

创建保存和取消按钮

保存和取消按钮都得有让模式视图消失的功能，所以我们首先让它们可以做到这一点。我们需要创建一些动作方法，以及按钮自身。我们已经讲过了如何用Interface Builder来做这些，这次我们要用代码来直接编程完成了。

```
- (IBAction) save: (id) sender;
- (IBAction) cancel: (id) sender;
```

这些就放在@end前面。

AddDrinkViewController.h

由于我们使用了导航栏，所以自带了对左右两侧按钮的支持。我们只要创建出这些按钮，再将其赋给leftBarButtonItem和rightBarButtonItem属性，就可以让它们按照我们的要求各就各位了。

正如制造加号按钮时一样，我们打算使用导航控制器的左右按钮来进行保存和取消操作。

这次我们会用代码来构造按钮。

```
- (void)viewDidLoad {
      [super viewDidLoad];
      self.navigationItem.leftBarButtonItem = [[[UIBarButtonItem
alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemCancel
target:self action:@selector(cancel:)] autorelease];
      self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem
alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemSave
target:self action:@selector(save:)] autorelease];
```

AddDrinkViewController.m

注意这里的"autorelease"——通常我们分配一个类，会把它赋给需要的地方，然后释放对它的引用。通过在创建时使用自动回收机制，可以让Objective-C稍后替我们处理这些释放工作。与我们自己显式地处理这些相比，效率要差一点，但是代码看起来更整洁。

【拼吾爱-最新编程资源的分享下载站】

编写保存与取消动作方法

当用户点击Save或者Cancel时，我们需要退出模式视图，这是通过请求呈现该视图的视图控制器，对其做释放操作而实现的。然而，为了能更简便一些，我们可以向模式视图发送这个释放消息，而它则会自动将这个消息转发给自己的父视图控制器。由于AddDrinkViewController是模式视图，而且拥有按钮的回调方法，我们只要向自身发送释放消息就可以了，控制器堆栈会对其做出正确处理。我们需要向自身发送`dismissModalViewControllerAnimated:`消息，如下所示：

```
[self dismissModalViewControllerAnimated:YES];
```

用下面的代码写出保存与取消方法，在被调用时写日志，然后清除模式视图。一旦这些可以工作了，我们就会实际进行保存饮料的工作。

把这些放在文件的末尾，刚好在dealloc的前面。

```
#pragma mark -
#pragma mark Save and Cancel

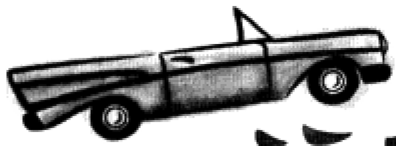
  - (IBAction) save: (id) sender {
      NSLog(@"Save pressed!");
      [self dismissModalViewControllerAnimated:YES];
}
  - (IBAction) cancel: (id) sender {
      NSLog(@"Cancel pressed!");
      [self dismissModalViewControllerAnimated:YES];
}
```

因为我们是模式视图，这个释放消息会被上交到父视图控制器，后者会实际完成让视图消失的工作。

AddDrinkViewController.m

现在，看看按钮是否工作了……

【拼吾爱-最新编程资源的分享下载站】



运行测试

现在模式视图可以被打发走了，键盘也工作正常了！







热烈祝贺，模式视图工作正常！

在本章中，你学到了如何添加视图，并将其传递给导航堆栈来弹出视图，以及如何重用已创建好的nib文件，并将其调整用于新的用途！不仅如此，你添加的视图是模式视图，你还可以释放视图，令其消失。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问： 为什么我们不需要给保存/取消按钮弄一个输出口？Interface Builder怎么办？

答： 导航控制器API支持左右两侧按钮；你只需将它们设置成你要用的按钮就行了（例如，保存与取消按钮）。做完这些之后，剩下的工作就只有将其与动作方法匹配了。

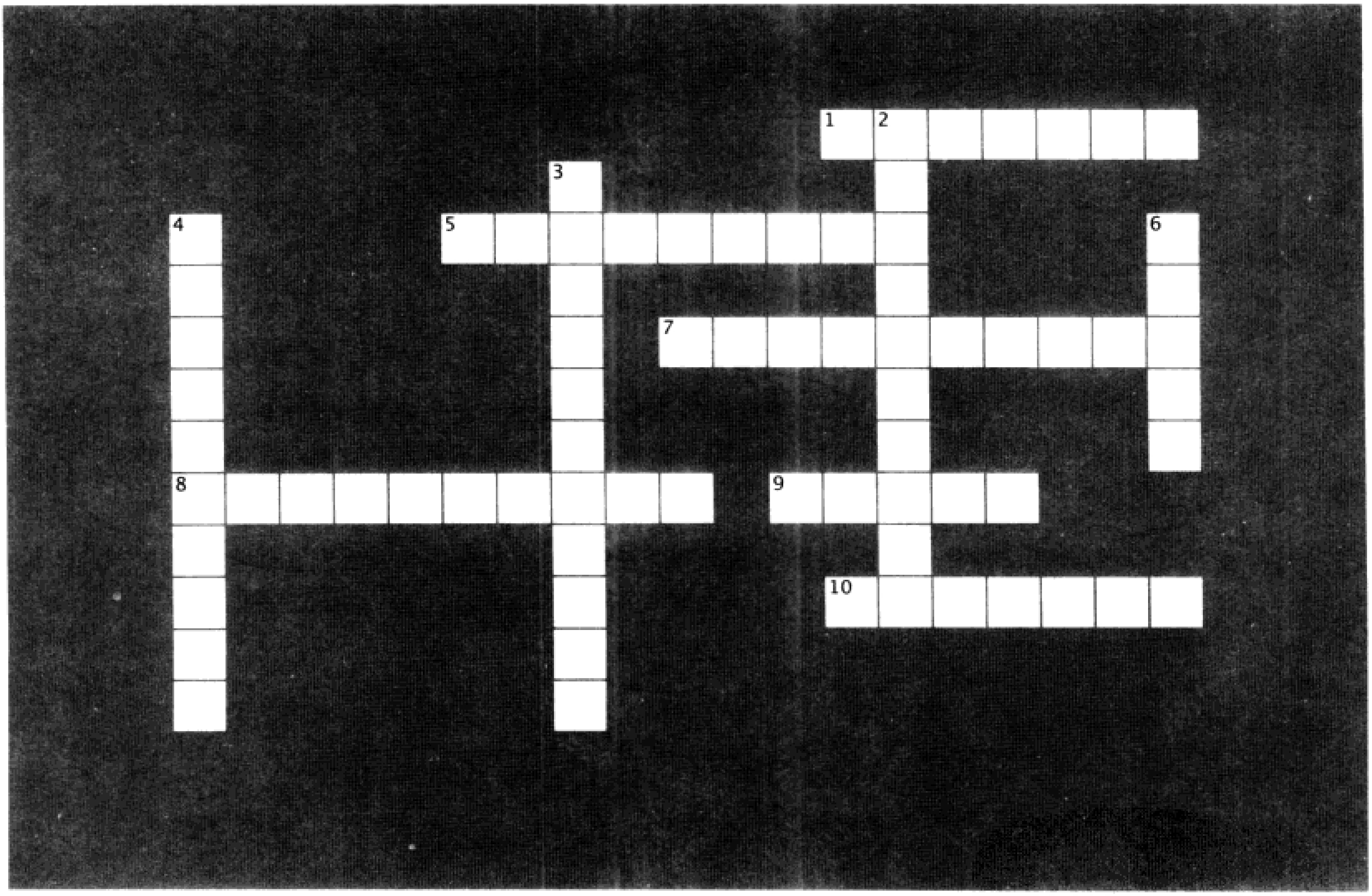


未完待续……

【拼吾爱-最新编程资源的分享下载站】

iPhone开发纵横字谜

用你学到的所有处理各种plist和视图的内容，填进下面的字谜中……



横向提示

1. 导航控制器支持_________按钮，可以对内容进行修改。

5. 用它们来对事物的名称进行组织和管理。

7. 视图可以被_________以进行扩展，与其他任何类一样。

8. 你可以在IB中，也可以用代码来创建_________栏。

9. 如果UI与行为分离的话，_________会更容易。

10. 用户在iTunes上的_________，会一直伴随应用，即使发布了新版本之后仍然如此。

纵向提示

2. HIG要求，如果还可以获得进一步的信息时，单元格中必须有某种类型的_________元素。

3. ________指定了按钮看起来应该是什么样子。

4. 一个nib文件中包含有UI________。

6. 用户必须首先处理________视图，然后才能做其他的事。

【拼吾爱-最新编程资源的分享下载站】



你的iPhone工具箱

你已经消化吸收了第5章的内容，现在可以把plist和模式视图加入到你的工具箱中了。如果想得到一份完整的书中功能提示的列表，请访问http://www.headfirstlabs.com/iphonedev。

调试

如果你知道问题大概出自何处，可以在那里设置断点。

你可以用调试器单步执行有问题的地方。

如果不知道从何入手，可以单步跟踪整个应用！

iTunes常识

1. 将应用提交到应用商店就一定要遵循HIG。

2. 审查通过可能要好几周时间，所以最好应用没有问题，一次通过。

3. 一旦你的应用上架发售，评论就不会消失了，即使你更新了应用也一样。

字典

是一种很强大的扩展plist内容的方法。

应用需要正确地对其进行处理。

视图

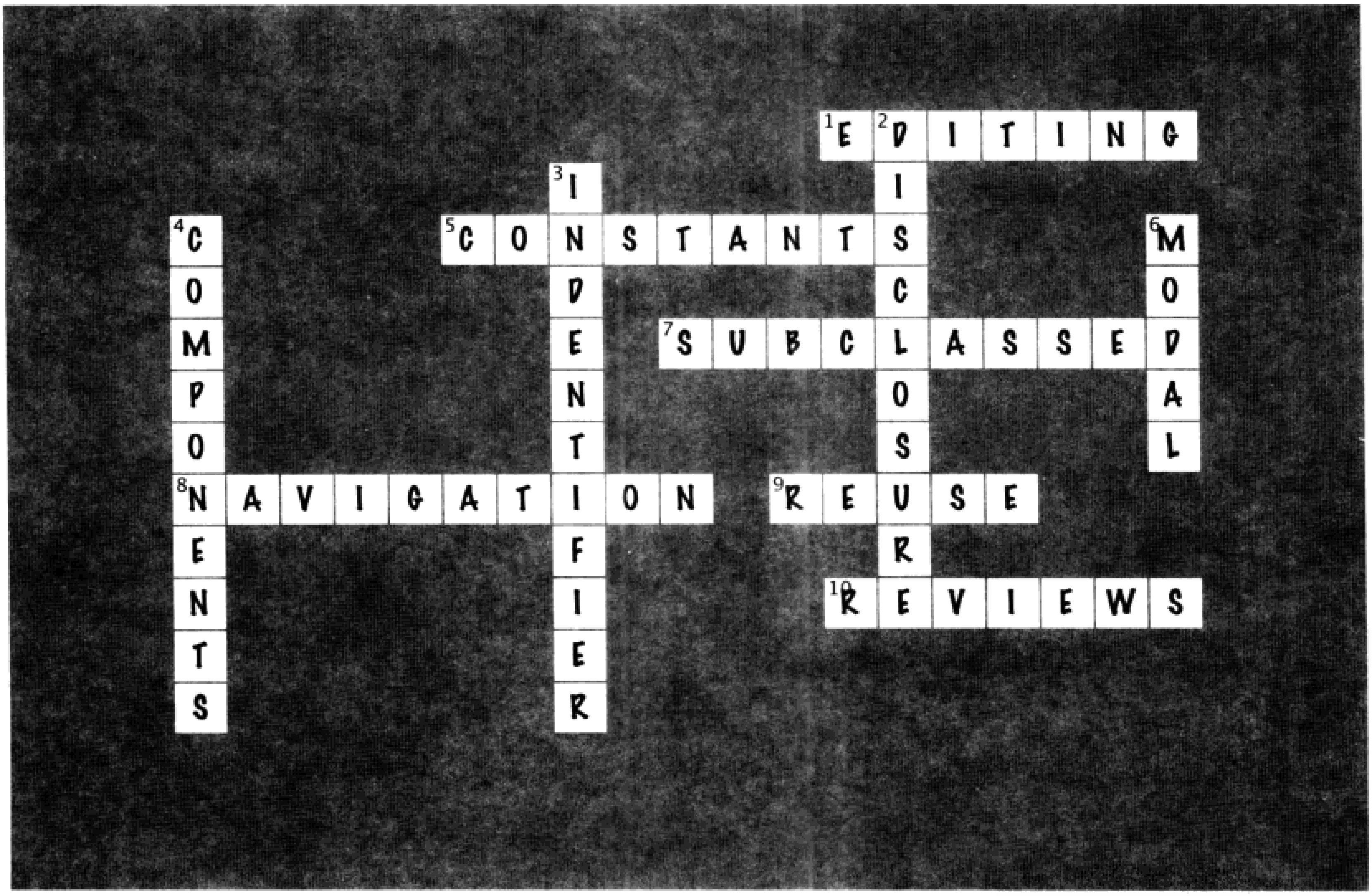
可以通过表格视图或者按钮压入视图堆栈中。

象其他类一样，可以通过继承派生出子类进行扩展。

模式视图强制用户要与其进行互动，然后才会消失。

【拼吾爱-最新编程资源的分享下载站】

iPhone开发纵横字谜答案



横向提示

1. 导航控制器支持________按钮，可以对内容进行修改。[EDITING]

5. 用它们来对事物的名称进行组织和管理。[CONSTANTS]

7. 视图可以被________以进行扩展，与其他任何类一样。[SUBCLASSED]

8. 你可以在IB中，也可以用代码来创建________栏。[NAVIGATION]

9. 如果UI与行为分离的话，________会更容易。[REUSE]

10. 用户在iTunes上的________，会一直伴随应用，即使发布了新版本之后仍然如此。[REVIEWS]

纵向提示

2. HIG要求，如果还可以获得进一步的信息时，单元格中必须有某种类型的________元素。[DISCLOSURE]

3. ________指定了按钮看起来应该是什么样子。[INDENTIFIER]

4. 一个nib文件中包含有UI________。[COMPONENTS]

6. 用户必须首先处理________视图，然后才能做其他的事。[MODAL]

【拼吾爱-最新编程资源的分享下载站】

6 数据的保存、编辑与排序

人人都能当编辑……



能显示数据挺不错的，但是添加和编辑信息才是让应用能够真正有价值的东西。 DrinkMixer很棒——它使用了一些单元格定制技术，而且使用plist字典来储存数据。它是一个很方便的参考工具应用，而你已经在添加新饮料方面开了一个好头。现在，是时候给用户以修改数据的能力了——保存、编辑以及排序——这会令这个应用对大家更有用处。在这一章中，我们会看一看iPhone应用中的编辑模式，以及如何通过导航控制器来引导用户。

【拼吾爱-最新编程资源的分享下载站】

山姆准备添加一种叫“红发学生妹”的饮料……

休闲吧的一种新饮料。

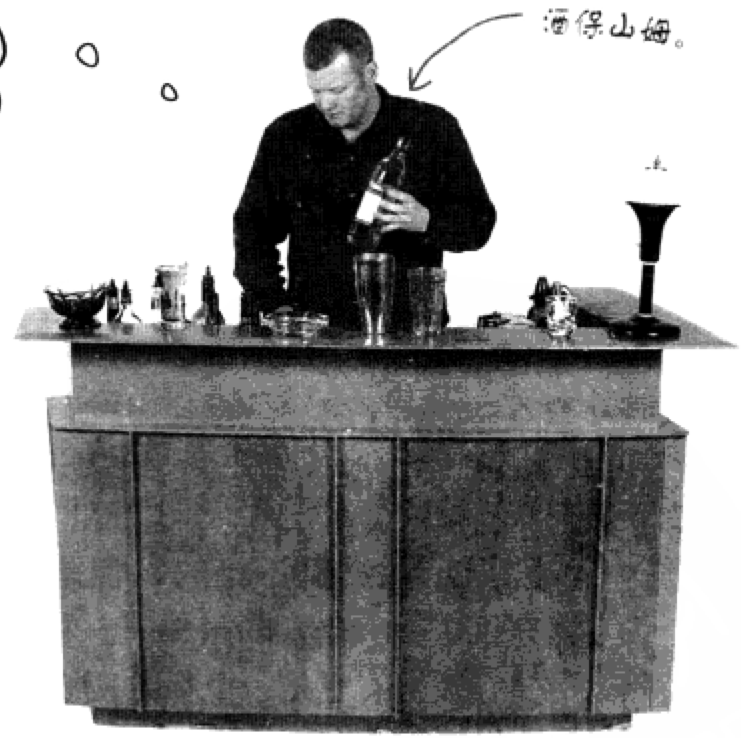
山姆试了一下DrinkMixer新增的添加饮料视图，马上就遇到了问题。



山姆四处点击，准备添加他的新饮料。

调制指南字段被键盘挡住了。

根本就没法看到调制指南，而且成份信息也有一部分被挡住了。



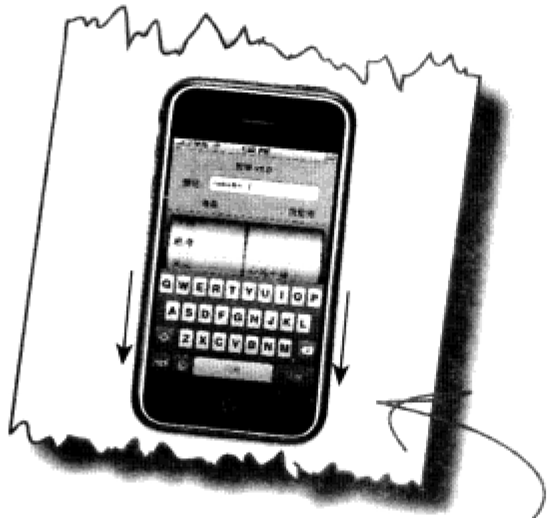
酒保山姆。

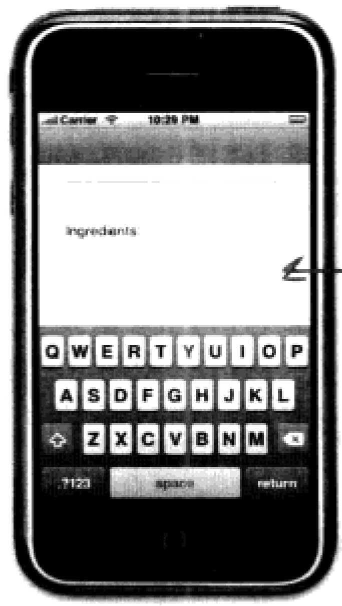
我们的视图有问题，因为有一些字段无法使用。

【拼吾爱-最新编程资源的分享下载站】

...但是键盘总是碍事

我们又回到早先在InstaTwit上碰到的那个键盘问题了。当山姆点击一个控件时，它就会获得焦点（成为第一响应者），并请求iPhoneOS显示键盘。通常这是一件好事。然而……





脑力举重



上次我们是怎么解决键盘问题的？这次同样的方法能管用吗？在键盘出现时你想让视图怎么做？

....

....

....

....

【拼吾爱-最新编程资源的分享下载站】

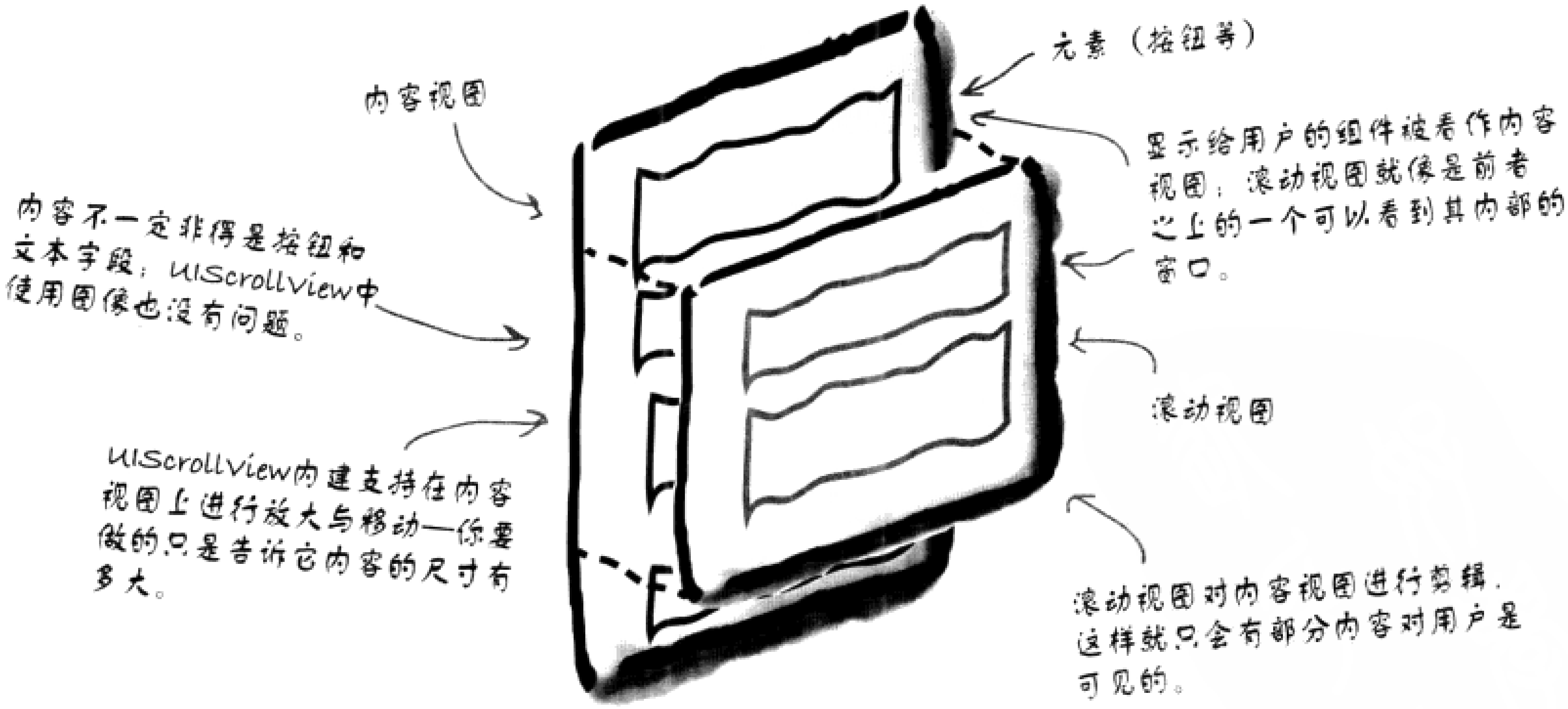
脑力举重

上次我们是怎么解决键盘问题的？这次同样的方法能管用吗？在键盘出现时你想让视图怎么做？

上次退出第一响应者状态就行了。对于DrinkMixer，这样做对名称字段应该可行，但是调制指南和成份字段怎么办？键盘一出现，它们就会被覆盖。只要键盘显示出来，用户就只有一小块屏幕可用——我们得在用户需要时将视图内容滚动进画面。我们可以用UIScrollView做到这一点。

UIScrollView全接触

UIScrollView与我们一直在用的基本UIView类似，除了一点，它里面的内容（比如按钮、文本字段等）可以被放置在屏幕以外，然后再滚动进入画面。滚动视图会绘制一个滚动条，并且可以通过移动和放大显示内容，来控制视图中哪些部分被显示出来。它完成这些工作的前提是知道需要显示的区域大小（即`contentSize`），以及需要在其中显示的内容占据的空间（`frame`）。然后UIScrollView就可以自己推算出剩下的全部信息。



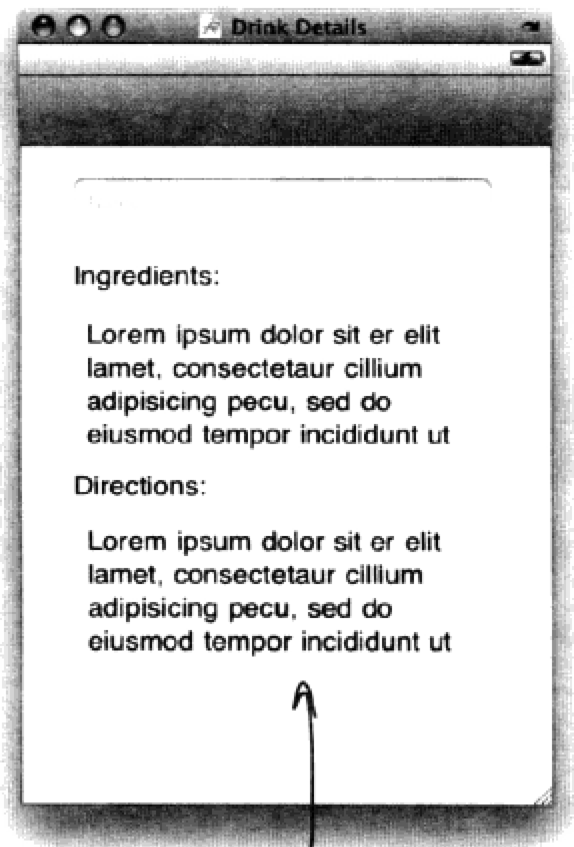
请记住，在CocoaTouch中组件都是UIView的子类。一个滚动视图需要注意的就是自己要管理的子视图而已。这个子视图是什么无关紧要，它可以是一个巨大的UIImageView，显示了一幅很大的图片，也可以是许多文本字段、按钮和标签。

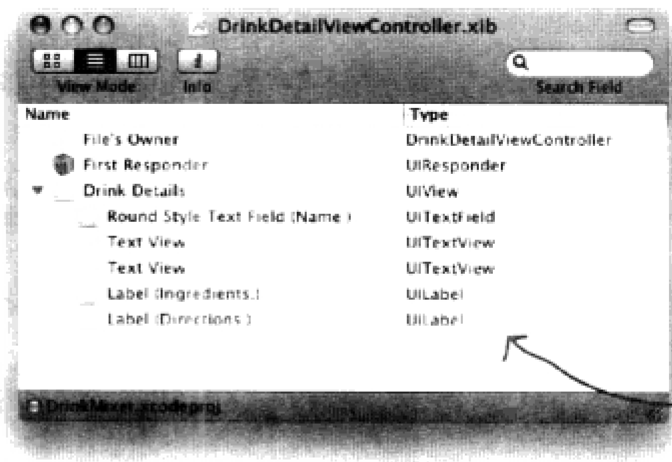
要得到一个滚动视图，我们需要将组件从UIView中移到一个UIScrollView里面。现在该是回到Interface Builder的时候了……

【拼吾爱-最新编程资源的分享下载站】

我们需要将内容包装到一个滚动视图中

我们要让用户可以在键盘覆盖某些控件时，能够对视图进行滚动，将其显示出来。为了达到这个目的，我们需要在视图上添加一个UIScrollView，并令其了解要处理哪些控件（内容视图）。





这些组件全都需要成为滚动视图的子节点。

滚动视图大小将是整个视图（减去导航栏）

滚动视图现在需要持有这3个组件。

这可真烦人。你的意思是我们必须把这些组件全都去掉，然后再重新布置整个视图？没有简单一点的办法吗？



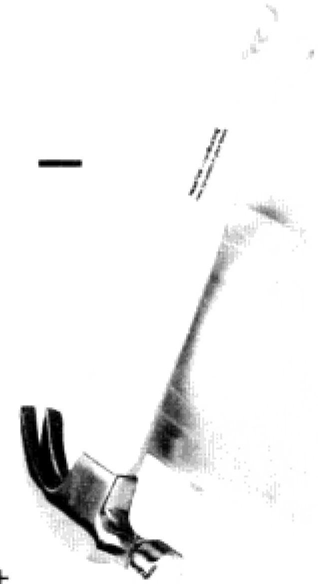
你说到点子上了。

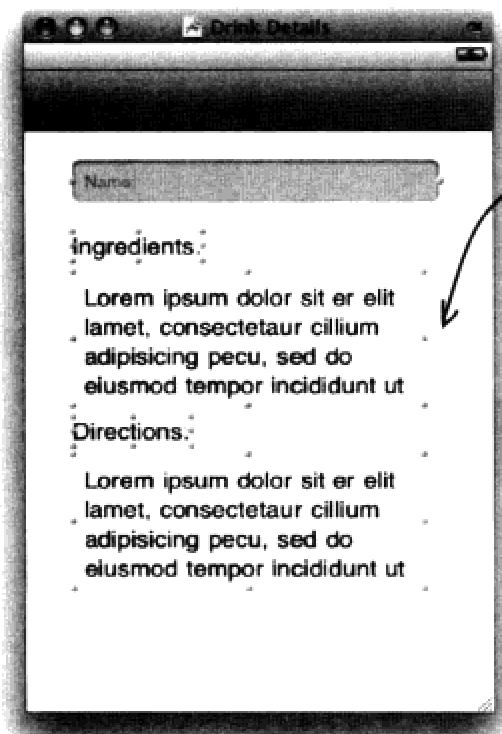
记得吗？我们说过有些时候Interface Builder可以让事情变得容易一些（很多）。这就是其中一种情况……

【拼吾爱-最新编程资源的分享下载站】

轻松打造GUI

很显然，将视图打造完毕之后又意识到需要让它能够滚动，并不是我们的独家专利。对于将一个现有的视图包装进一个UIScrollView中，Interface Builder中有内置的支持。

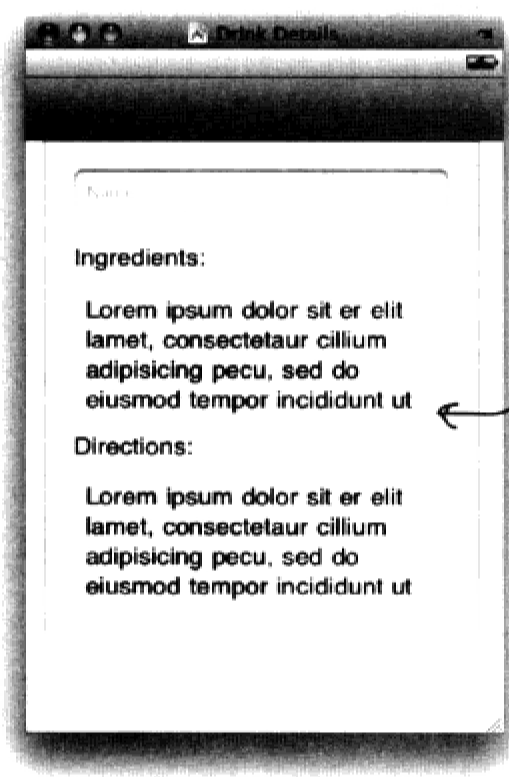




在细节视图中选中所有控件（如图所示），然后在菜单选项中依次选择**Layout → Embed Objects In → Scroll View**。Interface Builder就会自动创建一个新的滚动视图，并将所有控件固定在滚动视图中同样的位置上。



你的控件列表和以前一样，但是现在它们是在滚动视图下面。



Interface Builder会创建一个大小正好能装得下所有组件的UIScrollView。由于我们希望整个视图都能滚动，所以要拽住新的UIScrollView的角，把它向某个方向拖出屏幕一些，上面还是要齐着导航栏的边缘（我们不想让它也滚动）。

脑力练习

这个新的滚动视图怎么会知道有多少内容需要被滚动呢？

【拼吾爱-最新编程资源的分享下载站】

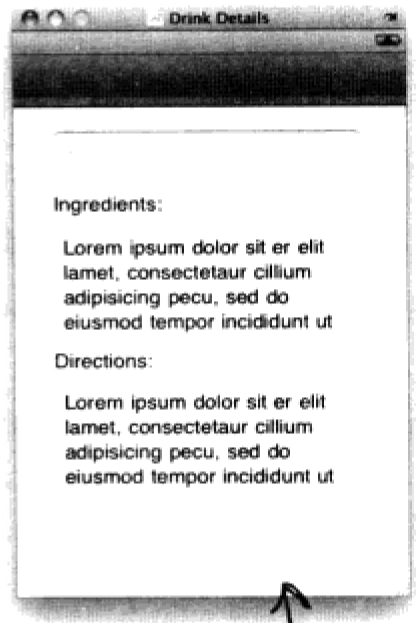
滚动视图大小与屏幕相同

Interface Builder创建了UIScrollView，但是还剩下最后一些收尾工作必须手工完成，才能让它按照我们的要求运转。我们需要告诉UIScrollView内容区域有多大，这样它才知道需要滚动哪些部分。我们通过设置它的contentSize属性来完成此项工作。你还得给UIScrollView添加对应的属性和输出口，然后再于Interface Builder中将它们联系起来，这样我们才能用得到它。

那么，怎样才能得知contentSize应该是多大呢？如果UIScrollView与屏幕的尺寸相同，我们没有任何内容是位于可见区域之外的，它根本不用操心如何滚动内容。由于滚动视图的大小与我们放进去的UIView是一样的，所以可以从这里获取大小，如下所示：

```
scrollView.contentSize = self.view.frame.size;
```

当你加上这一行代码之后，你得到的滚动视图就会在占据所有的空间的同时，认为自己的内容视图也是同样大小。



当你改变它的大小之后，UIScrollView和它的contentSize大小就是相同的了。我们只需要将这一点告诉滚动视图就行了。

拿起笔来

更新DrinkDetailViewController.h和DrinkDetailViewController.m文件，加入对新增UIScrollView的处理。

1. 为DrinkDetailViewController添加一个名为scrollView的属性，存放对UIScrollView的引用。你需要声明成员变量，将其标志为IBOutlet，然后在.m文件中合成属性，并在dealloc中将其释放。

2. 在Interface Builder中添加一个新的引用输出口，将新建的scrollView属性与UIScrollView联系起来。

3. 在viewDidLoad:方法中，为scrollView设置初始的contentSize。记住，我们要告诉scrollView，它的内容的大小与放在它里面的视图一样大。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

更新DrinkDetailViewController.h和DrinkDetailViewController.m文件，加入对新增UIScrollView的处理。

1 为DrinkDetailViewController添加一个名为scrollView的属性，存放对UIScrollView的引用。你需要声明成员变量，将其标志为IBOutlet，然后在.m文件中合成属性，并在dealloc中将其释放。

```
...
@interface DrinkDetailViewController : UIViewController {
      NSDictionary *drink;
      IBOutlet UITextField *nameTextField;
      IBOutlet UITextView *ingredientsTextView;
      IBOutlet UITextView *directionsTextView;
      IBOutlet UIScrollView* scrollView;
}

@property (nonatomic, retain) UIScrollView* scrollView;
...
```

为新加入的scrollView增加一个属性和IBOutlet的声明。

DrinkDetailViewController.h

合成属性，然后在viewDidLoad方法中设置contentSize。

```
@synthesize scrollView;
```

```
-   (void)viewDidLoad {
     [super viewDidLoad];
     scrollView.contentSize = self.view.
frame.size;
}
```

设置初始的contentSize值。

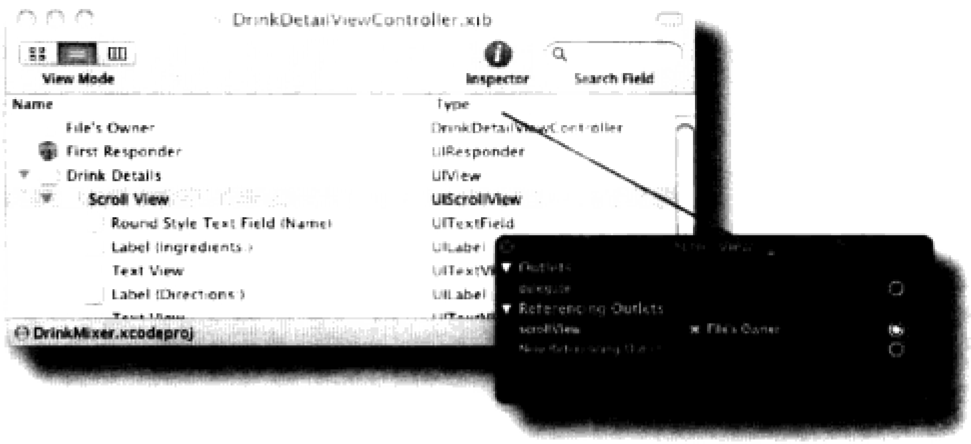
在dealloc里清理引用。

```
- (void)dealloc {
        [scrollView release];
        [nameField release];
        [ingredientsTextView
release];
...
```

DrinkDetailViewController.m

【拼吾爱-最新编程资源的分享下载站】

❷ 在Interface Builder中将新建的属性与UIScrollView联系起来。



运行测试



在文本字段中点击，键盘就会出现……但是没有任何东西滚动起来！

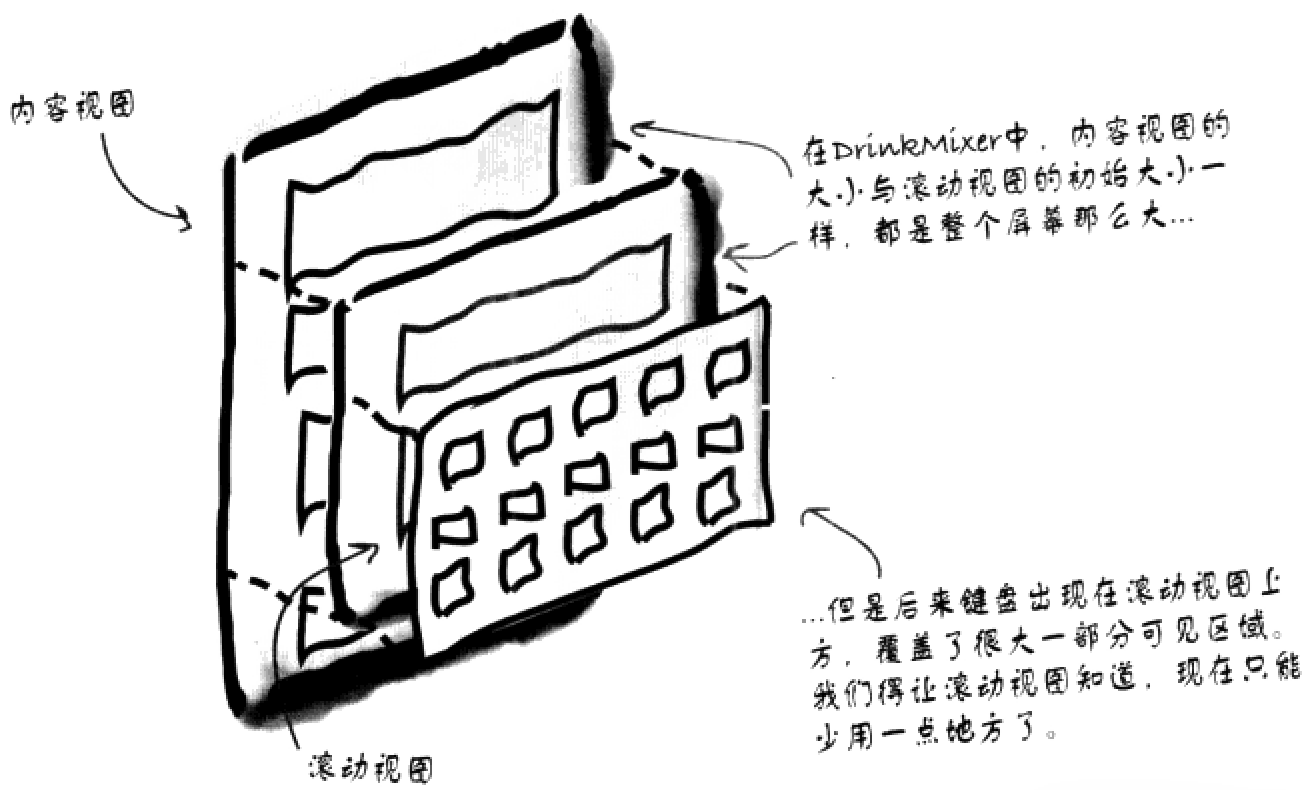
脑力练习

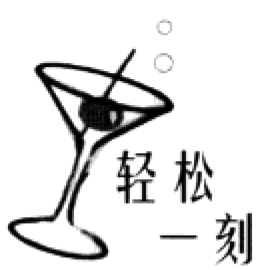
为什么还是不管用？想一想你放进这个视图里面的都有哪些东西——滚动视图、主视图以及键盘……

【拼吾爱-最新编程资源的分享下载站】

键盘改变了可见区域

问题在于键盘会改变可见区域，但是滚动视图对发生的一切一无所知。滚动视图仍然认为可以用整个屏幕来显示自己的内容，在它看来，空间还很多呢。我们需要告诉滚动视图，可见区域现在变小了，因为键盘出现了。





iPhone会告诉你键盘的事，但是不会修正你的视图。

这是因为iPhone知道键盘出现了，但是它不知道你的应用想怎么处理这个情况。这完全取决于你！

【拼吾爱-最新编程资源的分享下载站】



【拼吾爱-最新编程资源的分享下载站】

iPhone会通知你有关键盘的消息

与键盘和滚动视图互动的过程，带领我们来到了iPhone OS的一个之前从未被提起过的领域，名叫通知。正如组件事件会在应用程序之中传递一样，也存在着系统层次的事件，这就是通知，它们在iPhone OS中四处传递。要了解关于键盘的情况，关键就在于这些通知事件。



1 山姆在饮料名称字段中点击，该字段变为第一响应者。现在iPhone OS需要将键盘显示出来。

2 iPhone OS向默认的NSNotificationCenter发布一个名为UIKeyboardDidShowNotification的通知。

事件	对象	选择子
UIKeyboardDidShowNotification	DetailDrinkViewController	keyboardDidShow

NSNotificationCenter

3 NSNotificationCenter查找是否有人注册了这个事件。对象注册时要提供一个选择子（方法），如果事件被触发，就会调用这个方法。

4 NSNotificationCenter调用目标选择子，并将关于事件的信息传递给它，包括触发事件的对象以及事件的特定细节。

```
[registeredObject
keyboardDidShow:eventInfo];
```



【拼吾爱-最新编程资源的分享下载站】

向默认通知中心订阅事件

iPhone OS支持多个NSNotificationCenter，但是除非你有特殊的需求，需要自己建立一个，否则使用默认的系统级别的中心就可以了。你可以通过下面的调用，得到对默认中心的一个引用：

```
[[NSNotificationCenter defaultCenter];
```

有了通知中心，你就可以注册订阅事件了，注册时要传递你希望让通知中心回调的对象（通常就是自己）、调用的方法、你感兴趣的事件（或者用nil注册所有事件）以及一项可选的内容——希望监听的发送者（nil表示所有发送者）。

由于我们对系统通知感兴趣，所以我们使用默认通知中心。

我们希望通知中心调用我们(DetailDrinkViewController)，所以将self传进去做观察者。

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@
selector(keyboardDidShow:) name:UIKeyboardDidShowNotification object:nil];
```

从方法名创建一个选择子，就像动作方法一样。

不要忘记这里的冒号，因为你会通过一个参数得到通知的细节。

由于我们仅仅在自己的窗口可见时才订阅键盘事件，所以不在乎是谁发送了事件。

用完之后再取消订阅

正如内存管理一样，当我们不需要继续获取通知的时候，也要在通知中心中将注册清理掉。我们订阅的事件是`viewWillAppear:`和`viewWillDisappear:`。取消对事件的订阅很简单——只要请求通知中心用**removeObserver**将你注册的对象删除即可。

```
[[NSNotificationCenter defaultCenter] removeObserver:self];
```

一定要确定你取消订阅的通知中心与注册的中心是同一个。

我们请求通知中心，将我们注册的一切都删除。如果你只想停止接收特定通知，也可以指定这些通知。

【拼吾爱-最新编程资源的分享下载站】



选取器揭密

本周访谈：
Why do you talk so much?

Head First: 嗯，虽然有点尴尬，我还是得说，我不是十分确定我们邀请到了正确的通知中心……

通知中心： 哦，除非你的需求比较怪异，否则应该就是我。我就是大家默认情况下会找的那个家伙。注意！一个应用退出了。马上就让您知道。

Head First: 哇——这么说每个应用启动停止你都知道啊？

通知中心： 正是。我是默认中心；所有系统事件都从我这儿过。如今大家对时事都不怎么关心了，不过如果谁想要知道，就得来找我。

Head First: 那么当有人想要了解最新消息，他们会告诉你自己对什么东西感兴趣，对吗？

通知中心： 完全正确。如果谁想要知道系统里的什么事，就得在我这儿注册。他们得告诉我要注意什么通知，事情出现了应该告诉谁，还有，要是他们真的很挑剔的话，还得指定事件是谁发出的。

Head First: 那当通知出现的时候，你就会告诉他们？

通知中心： 是的——他们会告诉我，当我看到他们感兴趣的通知的时候应该发送什么消息给他们。我会把通知的信息打包到一个对象中，然后调用他们的方法，同时把这个对象传给他们。这根本不费我什么功夫；发送者几乎总会等我告诉完所有人发生了什么之后，才会去做其他事情。

采访者： 几乎总会？

通知中心： 哎呀，发送者也可以通过一个通知队列，让我稍后等它不太忙的时候，再将通知发送出去。不过通常都不会这样做。

Head First: 嗯,听起来很像消息传递啊。发送者想要告诉某人发生了某件事情，你就调用那个人的一个方法……其中有什么不同吗？

通知中心： 这与消息传递类似，但是也有一些区别。首先，发送者不需要知道要告诉谁。他们只是告诉我某些事情发生了，而由我来找出是否有人感兴趣。其次，可能会有很多人对于发生的事件感兴趣。在一般的消息传递中，发送者必须单独告诉每一个人。用通知的话，他们只要告诉我一次就行了，我会确保让所有人都知道。最后，通知的接收者可以不在意是谁发送的消息。如果一些对象要得知应用程序正在关闭，他不用关心由谁来负责发送出应用退出的消息，只要信任我，我会在事情发生时让他们知道。

Head First: 那是随便什么人都能发送通知吗？

通知中心： 当然了。任何人都可以请求我发布一则通知，而如果有人注册了这个通知的话，我会负责让他们知道。

Head First: 他们怎么知道要发送什么通知呢？

通知中心： 啊，这完全取决于发送者。不同的框架都有自己不同的消息用来四处传递，你必须看看框架会发送什么通知。如果你打算发布自己的通知的话，几乎肯定不想把其他某个人的通知冲掉；所以你应当用自己定义的。这些通知就是一些字符串——如果你想要包含一些额外的信息的话，就是字典——没什么特殊之处。

Head First: 我明白了。好了，今天的访问很愉快，通知中心。谢谢你的光临！

【拼吾爱-最新编程资源的分享下载站】

拿起笔来

在下面的空白处填上下一步的计划！

我们需要在 ______________ 中， ______________

______________ 与 ______________ 事件。我们

会添加两个 ______________ ，以便让 ______________ 在通知发布时调用。

我们会在键盘出现与消失的时候，调整 ______________ 的大小。

我们需要在 ______________ 中， ______________ 事件。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

现在你制定出了下一步的计划。

我们需要在 viewWillAppear 中，注册订阅 UIKeyboardDidShowNotification 与 UIKeyboardDidHideNotification 事件。我们会添加两个 方法，以便让 通知中心 在通知发布时调用。

我们会在键盘出现与消失的时候，调整 滚动视图 的大小。

我们需要在 viewWillDisappear 中，取消订阅 事件。

有问必答

问：我无法找到iPhone OS所发送通知的列表。在哪里有这样的列表啊？

答： 不存在一个可能被发送的所有通知的集中列表。不同的类和框架会使用不同的通知。例如UIDevice类提供了一组通知，可以告诉你电池何时在充电以及近距离传感器的情况。苹果公司的文档通常都清楚地描述了可用的通知以及它们的含义。键盘通知是在UIWindow类的文档里描述的。

问：为什么我要创建自己的通知？

答： 这取决于你的应用程序。记住，通知让你可以解耦发送者与接收者。你可以在你的应用程序中，用这种方法让多个独立视图了解应用程序发生的事情。

例如，假如说你有一个视图，让你可以在应用程序中添加或者删除内容条目，而你的应用有几种不同的方式可以用来查看这些内容。那么通知就让你可以有一种很好的方法来向所有其他视图宣布有些东西被改变了，而不需要在添加/删除视图中保存对每个视图的引用。

【拼吾爱-最新编程资源的分享下载站】



练 习

行动起来，修改你的代码，去注册订阅键盘事件。
稍后我们会实现处理滚动视图的代码。

❶ **在AddDrinkViewController中添加keyboardDidShow和keyboardDidHide方法。**

目前，只要让它们在被调用时打印输出一条NSLog信息就行了。马上我们就会在里面加上实质内容。两个方法都接收一个`NSNotification*`参数，因为它们会被通知中心调用，参数用来传递通知信息。

❷ **在viewWillAppear(...)中注册UIKeyboardDidShowNotification与UIKeyboardDidHideNotification。**

你应该使用默认的NSNotificationCenter，注册对这两个事件的订阅接收，而不管是谁发送的通知。

❸ **在viewWillDisappear(...)中取消对所有事件的订阅注册。**

模板中带有一个此方法的存根，但是默认是被注释掉的。将注释去掉，并加入取消注册事件订阅的代码。

❹ **在AddDrinkViewController中添加一个BOOL成员，跟踪键盘是否可见的状态。**

马上我们就会进一步讨论这个问题，不过你会需要一个标志来跟踪键盘是否已经可见。目前先在viewWillAppear(...)里把它设置成NO。

【拼吾爱-最新编程资源的分享下载站】



练习
答案

行动起来，修改你的代码去注册订阅键盘事件。稍后我们会实现处理滚动视图的代码。

```
- (void)viewWillAppear: (BOOL)animated {
     [super viewWillAppear:animated];
     NSLog(@"Registering for keyboard events");
   [[NSNotificationCenter defaultCenter] addObserver:self selector:@
selector(keyboardDidShow:)
     name:UIKeyboardDidShowNotification object:nil];
[[NSNotificationCenter defaultCenter] addObserver:self selector:@
selector(keyboardDidHide:)
      name:UIKeyboardDidHideNotification object:nil];
     // 初始状态下键盘是隐藏的，所以将变量重置
     keyboardVisible = NO;
 }

 - (void)viewWillDisappear:(BOOL)animated {
     NSLog(@"Unregistering for keyboard events");
   [[NSNotificationCenter defaultCenter] removeObserver:self];
 }
 - (void)keyboardDidShow:(NSNotification *)notif {
     NSLog(@"Received UIKeyboardDidShowNotification.");
}
 - (void)keyboardDidHide:(NSNotification *)notif {
     NSLog(@"Received UIKeyboardDidHideNotification.");
}
```

这些都是实现文件中为键盘通知新添的方法。我们马上就会讲到它们。

我们需要跟踪键盘是否显示的状态。马上对此进一步展开。

如果你不指定要取消注册哪个通知，就会将你注册的所有通知都删除。

AddDrinkViewController.h

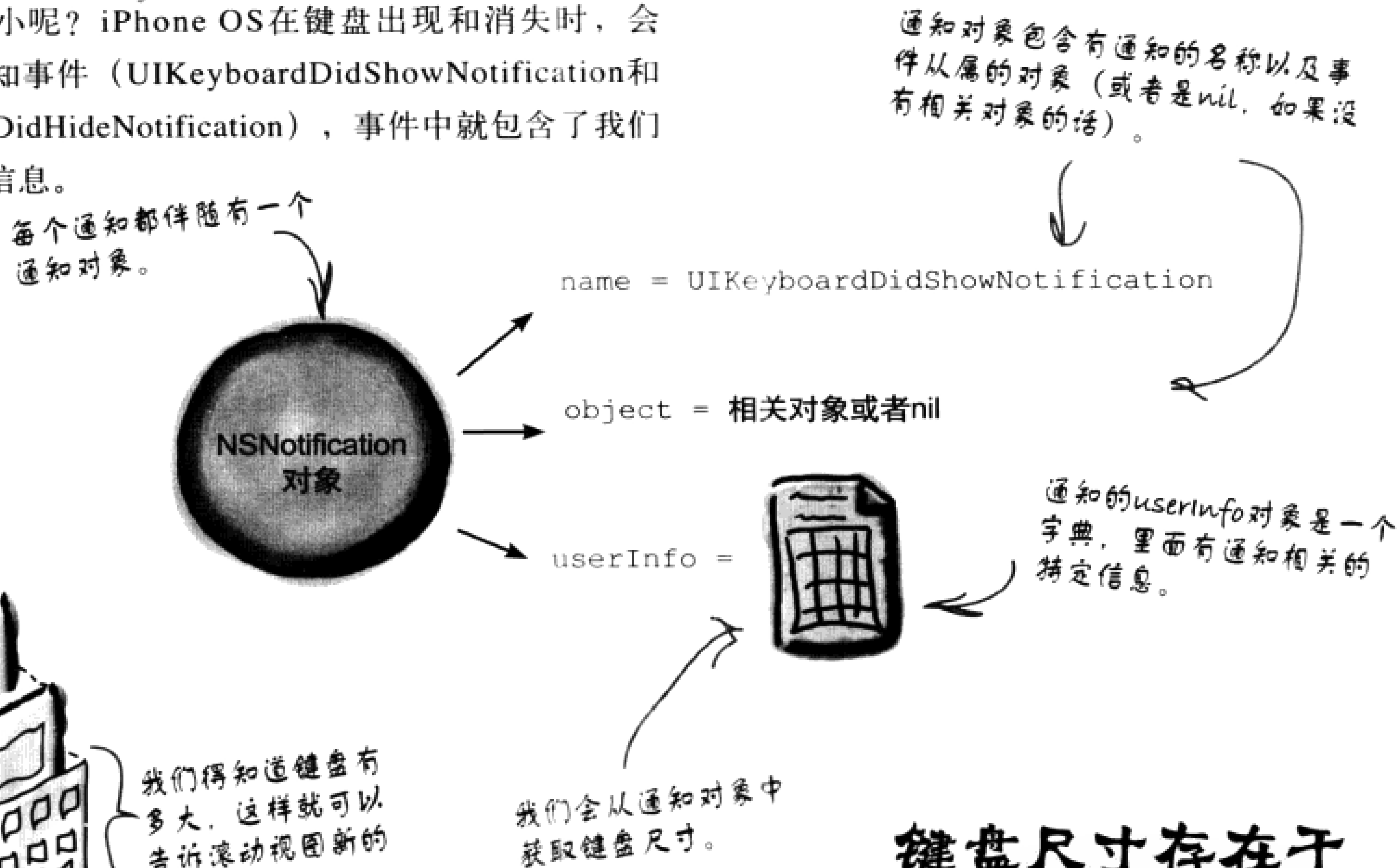
```
@interface AddDrinkViewController : DrinkDetailViewController {
     BOOL keyboardVisible;
}
- (void)keyboardDidShow: (NSNotification*) notif;
- (void)keyboardDidHide: (NSNotification*) notif;
}
```

AddDrinkViewController.h

【拼吾爱-最新编程资源的分享下载站】

键盘事件会告诉你键盘的状态和尺寸

了解键盘何时出现与消失的目的，完全在于让滚动视图知道可见区域的大小发生了改变。但是我们怎样才能得知新的大小呢？iPhone OS在键盘出现和消失时，会发送键盘通知事件（UIKeyboardDidShowNotification和UIKeyboardDidHideNotification），事件中就包含了我们所需的全部信息。





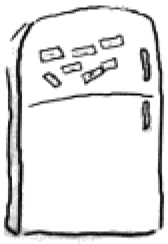
键盘尺寸存在于NSNotification对象中。



得到通知很简单，但是我们每次键盘显示的时候都会收到通知，甚至它已经出现的情况下也是如此。

这就是为什么我们要有一个BOOL变量来跟踪当前键盘是否显示。如果我们得到通知的时候键盘不可见，那么就需要告诉滚动视图可见尺寸变小了。如果键盘被隐藏起来，我们会将滚动视图的尺寸设置回完整大小。

【拼吾爱-最新编程资源的分享下载站】



键盘代码魔力贴第1部分

下面是一些代码魔力贴，你需要用它们来实现keyboardDidShow方法。请在下一页代码中注释的帮助下，找出每个魔力贴应该出现的位置。

```
CGRect viewFrame = self.view.frame;
viewFrame.size.height -= keyboardSize.height;
```

```
NSValue* aValue = [info objectForKey:UIKeyboardBoundsUserInfoKey];
CGSize keyboardSize = [aValue CGRectValue].size;
```

```
scrollView.frame = viewFrame;
keyboardVisible = YES;
```

```
NSLog(@"Resizing smaller for keyboard");
```

```
if (keyboardVisible) {
  NSLog(@"Keyboard is already visible.  Ignoring notification.");
  return;
}
```

```
NSDictionary* info = [notif userInfo];
```

【拼吾爱-最新编程资源的分享下载站】


```
- (void)keyboardDidShow:(NSNotification *)notif {




    // 之前键盘是不可见的


    // 获取键盘的尺寸。





    // 改变滚动视图的大小，为键盘腾出空间



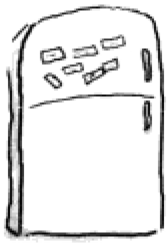


}
```



AddDrinkViewController.m

【拼吾爱-最新编程资源的分享下载站】



键盘代码魔力贴第2部分

下面是一些代码魔力贴，你需要用它们来实现keyboardDidHide方法。请在下一页代码中注释的帮助下，找出每个魔力贴应该出现的位置。

```
scrollView.frame = viewFrame;
keyboardVisible = NO;
```

```
NSLog(@"Resizing bigger with no keyboard");
```

```
if (!keyboardVisible) {
      NSLog(@"Keyboard already hidden.  Ignoring notification.");
      return;
}
```

```
NSDictionary* info = [notif userInfo];
```

```
NSValue* aValue = [info objectForKey:UIKeyboardBoundsUserInfoKey];
CGSize keyboardSize = [aValue CGRectValue].size;
```

```
CGRect viewFrame = self.view.frame;
viewFrame.size.height += keyboardSize.height;
```

【拼吾爱-最新编程资源的分享下载站】


```
- (void)keyboardDidHide:(NSNotification *)notif {




    // 键盘是可见的

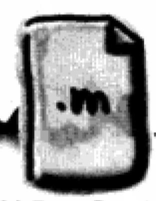
    // 获得键盘的尺寸。



    // 将滚动视图的高度重置为初始值




}
```



AddDrinkViewController.m

【拼吾爱-最新编程资源的分享下载站】

键盘代码魔力贴答案

下面是使用键盘要用到的代码魔力贴……

```
- (void)keyboardDidShow:(NSNotification *)notif {
    if (keyboardVisible) {
      NSLog(@"Keyboard is already visible.  Ignoring notification.");
      return;
    }

  // 之前键盘是不可见的
    NSLog(@"Resizing smaller for keyboard");

  // 获取键盘的尺寸。
    NSDictionary* info = [notif userInfo];
    NSValue* aValue = [info objectForKey:UIKeyboardBoundsUserInfoKey];
    CGSize keyboardSize = [aValue CGRectValue].size;

  // 改变滚动视图的大小，为键盘腾出空间
    CGRect viewFrame = self.view.frame;
    viewFrame.size.height -= keyboardSize.height;
    scrollView.frame = viewFrame;
    keyboardVisible = YES;
}
```

只要用户切换文本字段，我们就会收到通知，即使键盘已经显示了也是如此。所以我们跟踪它的状态，如果重复了就直接退出。

NSNotification带有一个字典，里面有事件的细节；这里我们将其取出。

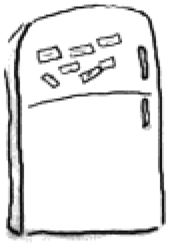
我们从字典中得到键盘尺寸……

……然后计算出滚动视图现在的实际大小（基本上就是视图大小减去键盘尺寸）。

最后，更新滚动视图的大小，并记录下来键盘已经显示。

AddDrinkViewController.m

【拼吾爱-最新编程资源的分享下载站】



键盘代码魔力贴答案第2部分

下面是使用键盘要用到的代码魔力贴……

处理UIKeyboardDidHideNotification的方式几乎完全一样，除了这次滚动视图需要按照键盘（现已消失）的尺寸进行扩展。

```
- (void)keyboardDidHide:(NSNotification *)notif {

    if (!keyboardVisible) {
        NSLog(@"Keyboard already hidden. Ignoring notification.");
        return;
    }

    // 键盘是可见的
    NSLog(@"Resizing bigger with no keyboard");

    // 获得键盘的尺寸。
    NSDictionary* info = [notif userInfo];

    NSValue* aValue = [info objectForKey:UIKeyboardBoundsUserInfoKey];
    CGSize keyboardSize = [aValue CGRectValue].size;

    // 将滚动视图的高度重置为初始值
    CGRect viewFrame = self.view.frame;
    viewFrame.size.height += keyboardSize.height;

    scrollView.frame = viewFrame;
    keyboardVisible = NO;
}
```

如果我们知道键盘不可见，就忽略这个通知。

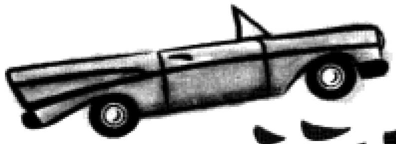
与以前一样，我们从事件中抽取键盘尺寸…

…并将滚动视图的尺寸重置为新的可见区域大小。



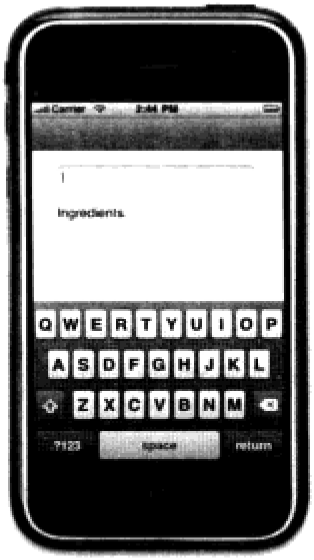
AddDrinkViewController.m

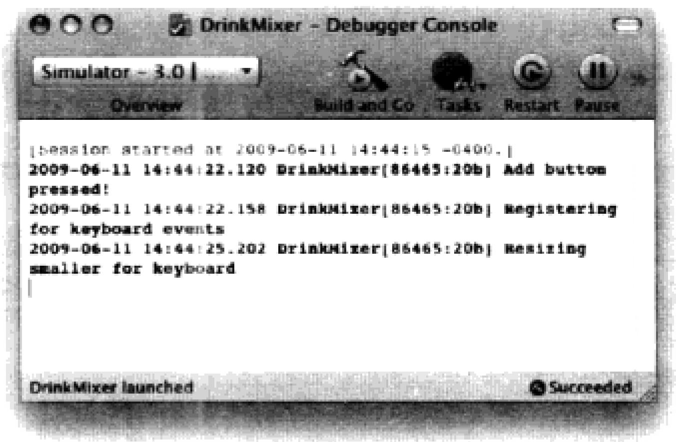
【拼吾爱-最新编程资源的分享下载站】



运行测试

继续，构建并运行。这次当你来到细节视图后，应该可以滚动这个视图，让它正确地显示字段了，而控制台中的消息可以帮助你了解所发生的事情。





有问必答

问：操纵滚动视图的大小有一些太技巧化了——要是没有魔力贴我怎么才能知道这样做呢？

答：《iPhone应用程序编程指南》是一个很好的参考资料，你可以在里面找到例子代码和通用的应用编程信息，在苹果公司开发者网站上可以找到这些内容。里面有针对常见问题的例子代码，比如处理键盘事件、使用GPS等。

问：能再说一次为什么要跟踪键盘是否已经显示的状态吗？这不应该是iPhone干的事儿吗？

答：iPhone OS知道键盘的状态，但是它会在不同控件获得焦点时发送键盘事件。所以当用户点击第一个字段时，你会得到一个UIKeyboard-WillShowNotification，然后紧接着一个UIKeyboardDidShowNotification。用户点击另一个字段时，你会得到另一个UIKeyboardDidShowNotification，这样你就知道键盘焦点变了，但并不会有键盘隐藏的消息，因为键盘实际上并未消失。你需要知道键盘是否已经显示了，这样才不会把滚动视图改成错误的大小。

问：滚动视图工作正常了，但是用户选择了不同的字段，是不是仍然需要将视图滚动到该控件处？

答：是的——这很不完美。如果你了解哪个控件拥有焦点，你可以让滚动视图滚动到内容视图的特定点。iPhone应用程序编程指南中有不错的例子代码。

问：我们真的需要从通知里得到键盘尺寸这些东西吗？不总是一样的吗？

答：不总是一样的！如果你的应用程序是横向的，键盘的宽度会大于高度。如果应用是纵向的，它的高度会大于宽度。苹果公司还声明有必要时会改变键盘的尺寸。你不应该对它的大小做出假设，一定要直接从键盘通知中获得尺寸信息。

【拼吾爱-最新编程资源的分享下载站】

一切都可以正常滚动了，而我也可以把饮料输入进去了，但是只要我一回到列表，输入的饮料就没了！



山姆的饮料丢了！

只要他一离开饮料细节视图，新饮料就没了，并不会显示在主列表中。我们需要找出办法，让它能存活更长时间……

拿起笔来

回答下列问题，考虑这些问题对于我们的应用的意义。

当用户点击保存时，对于新饮料发生了什么事儿？

....

....

我们该在哪里添加代码？

....

....

我们该怎样保存新饮料？

....

....

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

回答下列问题，考虑这些答案对于我们的应用意味着什么。

当用户点击保存时，对于新饮料发生了什么事儿？ 我们让视图消失，而饮料的信息也丢失了。

我们该在哪里添加代码？ 我们需要在保存方法中添加一些代码，以便实际保存用户输入的值。

我们该怎样保存新饮料？ 由于我们已经将饮料保存在字典中了，所以我们可以创建一个新字典存放这些信息，并将其添加到饮料数组中去。



我们需要给AddDrinkViewController加上一个引用，以便操作整个饮料数组。

创建一个新的NSMutableDictionary很容易，我们可以为其分配空间并进行初始化。我们也可以用setObjectForKey:来为字典设置饮料。稍微麻烦一点的事儿，是把它添加到饮料数组里去。我们可以在创建好RootViewController之后，让它将新饮料传递进去……

【拼吾爱-最新编程资源的分享下载站】

练习

回过头来修改RootViewController和AddDrinkViewController，以支持保存新饮料。

1. **给AddDrinkController添加一个对饮料主数组的引用。**

   你需要为这个类添加一个成员变量和一个属性，然后合成属性并在dealloc方法中释放引用。最后你需要确保RootViewController会在设置AddDrinkController的时候，传递进来一个对饮料数组的引用。

2. **创建一个新字典，并将其加入到数组中。**

   你需要修改save:方法，从控件中获得饮料细节，并将它们储存在一个新的字典中。然后，用`addObject:`将这个字典添加到饮料主数组中。

【拼吾爱-最新编程资源的分享下载站】

练习答案

回过头来修改RootViewController和AddDrinkViewController，以支持保存新饮料。

```
@interface
AddDrinkViewController : DrinkDetailViewController{
       BOOL keyboardVisible;
       NSMutableArray *drinkArray;
 }
@property (nonatomic, retain) NSMutableArray* drinkArray;
```

我们需要一个数组引用，以便稍后添加新饮料。

AddDrinkViewController.h

```
- (IBAction) addButtonPressed: (id) sender    {
       NSLog(@"Add button pressed!");

       AddDrinkViewController *addDrinkVC = [[AddDrinkViewController alloc] initWithNibName:@"DrinkDetailViewController" bundle:nil];
       UINavigationController *addNavCon = [[UINavigationController alloc] initWithRootViewController:addDrinkVC];
       addDrinkVC.drinkArray = self.drinks;
       [self presentModalViewController:addNavCon animated:YES];
       [addDrinkVC release];
       [addNavCon release];
}
```

将饮料主数组的引用交给新创建的AddDrinkViewController，好让用户能够添加新饮料。

RootViewController.m

【拼吾爱-最新编程资源的分享下载站】


```
#import "DrinkConstants.h"

@implementation AddDrinkViewController

@synthesize drinkArray;
```

我们需要键名常量，这样才能填写新字典。

我们需要合成新属性。

AddDrinkViewController.m

```
- (IBAction) save: (id) sender {
    NSLog(@"Save pressed!");

    // 创建一个新的饮料字典，来存放新值
    NSMutableDictionary* newDrink = [[NSMutableDictionary alloc] init];
    [newDrink setValue:nameTextField.text forKey:NAME_KEY];
    [newDrink setValue:ingredientsTextView.text forKey:INGREDIENTS_KEY];
    [newDrink setValue:directionsTextView.text forKey:DIRECTIONS_KEY];

    // 将其添加到饮料主数组中，然后释放我们的引用
    [drinkArray addObject:newDrink];
    [newDrink release];

    // 然后将细节视图弹出堆栈
    [self.navigationController dismissModalViewControllerAnimated:YES];
}
- (void)dealloc {
    [drinkArray release];
    [super dealloc];
}
```

由于我们要添加键和对象，所以得创建一个可更改的字典。如果你创建一个不可更改的版本，以后会遇到什么麻烦？

用键名常量添加饮料信息，然后把它附到饮料数组上去。

因为是我们分配的对象，所以也得由我们来释放。

在清理时，释放我们对drinkArray的引用。

AddDrinkViewController.m

【拼吾爱-最新编程资源的分享下载站】



妮柯，准备拍她的VIP客户的马屁。



预约失踪案件

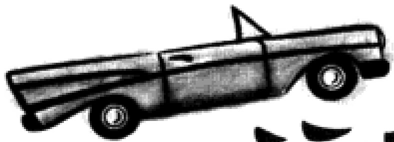
自从将近10年前鸭嘴兽之家开业起，妮柯就在这里做领班了。这所顶级餐厅的顾客中有许多杰出人士，他们喜欢这里完美的用餐体验。VIP客户名单已经有多年未曾更改了，而妮柯认得上面的每一个人。当VIP客户出现时，她会将他们领到最喜欢的座位，并确保一切都完美无误。她的工作极为高效，餐厅离开她就无法运转……直到最近，她犯了一个悲剧性的错误。

本月早些时候，鸭嘴兽之家有了一个新投资者。一位大名鼎鼎而又性情古怪的科学家、诺贝尔奖获得者，因其独特的品味而知名。餐厅管理层翻出了尘封的VIP名单，将这位科学家的大名列在了最后，并辅以详细指示，说明了当他光临时该如果确保万事“如常”。他们相信妮柯会打点好这位先生，对此没有一点怀疑。

昨天晚上，他们的新投资人光临了，比其他一些VIP顾客还早了几分钟。妮柯根本没注意到他，还是忙着领那些VIP常客到他们的座位，而在她的认知中，根本不存在什么新投资人。

为什么妮柯会忽视一位这么重要的新客人？

【拼吾爱-最新编程资源的分享下载站】



运行测试

这些代码可真不少啊！运行这个应用吧，确保一切工作正常。这里有一种饮料要加到列表里（这是Head First休闲吧的新看家饮料）。

把这个加到你的应用中。

红发学生妹

加拿大威士忌

奶油苏打水



在一个广口杯里加入威士忌，然后再加入奶油苏打水，并饮用。

【拼吾爱-最新编程资源的分享下载站】

运行测试

为了正确地测试这个应用，请点击加号按钮并在细节视图中输入新饮料的数据。做完以后，点击save进行保存。

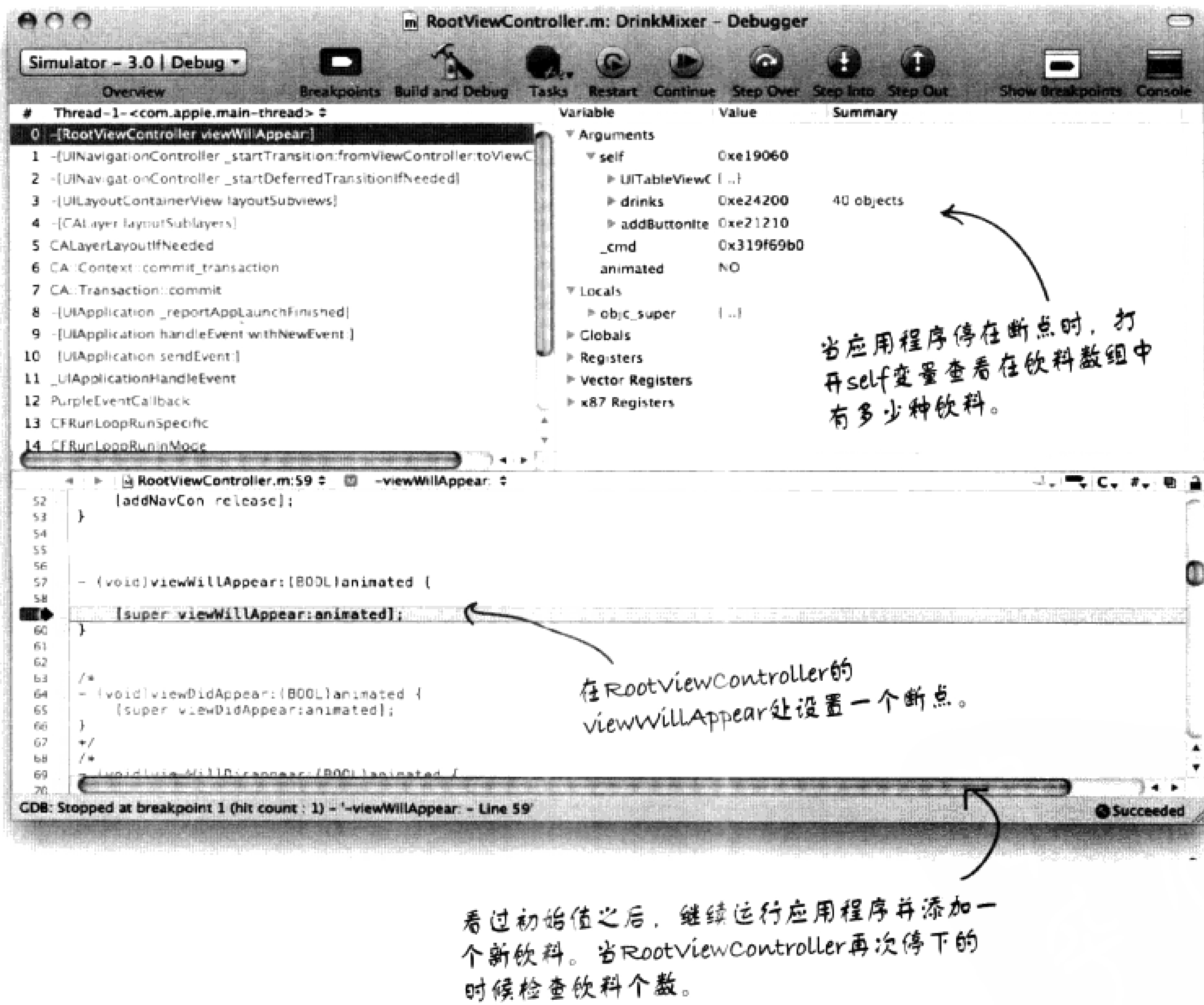
现在回到列表视图之后，发生了什么？



练习

调试

有些地方不对劲。我们实现了保存方法，创建了新饮料，添加到数组……而我们很肯定所有代码都工作正常。在继续行动之前，让我们用调试器来做个快速检查。去掉RootViewController.m中`viewWillAppear`的注释，并设置一个断点。点击“Build and Run”来启动应用程序……



你发现了什么？

....

....

发生了什么事？

....

....

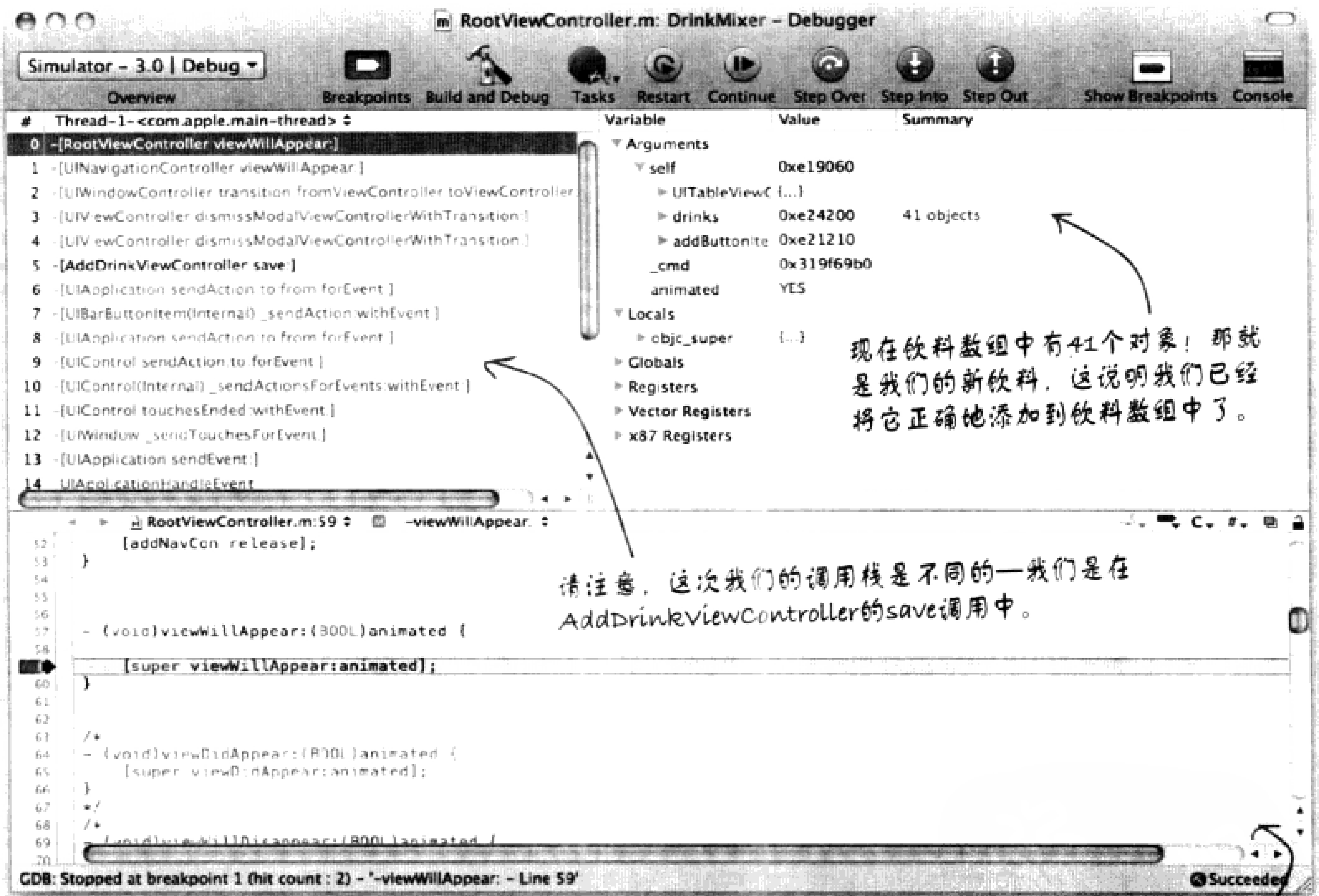
【拼吾爱-最新编程资源的分享下载站】



练习
答案

调试

现在我们要用调试器来帮我们找出发生了什么问题。



现在饮料数组中有41个对象！那就是我们的新饮料，这说明我们已经将它正确地添加到饮料数组中了。

请注意，这次我们的调用栈是不同的——我们是在AddDrinkViewController的save调用中。

*如果你想要看到每个字典中的name项，可以在控制台中使用下面这个命令：

```
p (char*)[[[self.drinks objectAtIndex:0] objectForKey:@"name"] UTF8String]
```

你发现了什么？ 数组里面开始有40个字典；添加了我们的新饮料之后，里面多了一个。我们用那条控制台命令可以遍历各项内容，会发现是对的。

发生了什么事？ 表格视图没有去取新饮料。我们已经把它添加到饮料数组，但是还没有加到实际的视图里去。看起来表格视图并不知道它的存在。

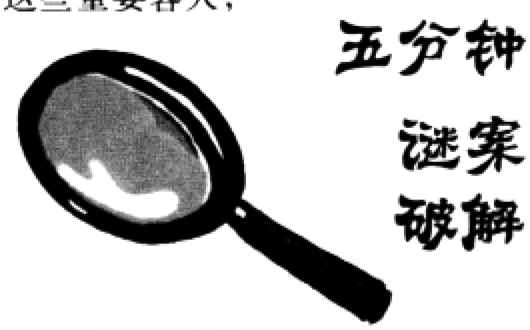
【拼吾爱-最新编程资源的分享下载站】



预约失踪案件已破获

为什么妮柯会忽视一位这么重要的新客人？

妮柯已经多年不用去查看VIP名单了。她非常在意这些重要客人，希望他们有宾至如归的感觉，所以她不想每次有人光临时都去粗鲁地查名单与人对照。她成功地记住了整个名单，这样当VIP来到餐厅时，她能认出他们，立即为他们安排座位。按照妮柯所知，名单上有10位VIP客户，她对他们都很熟悉。

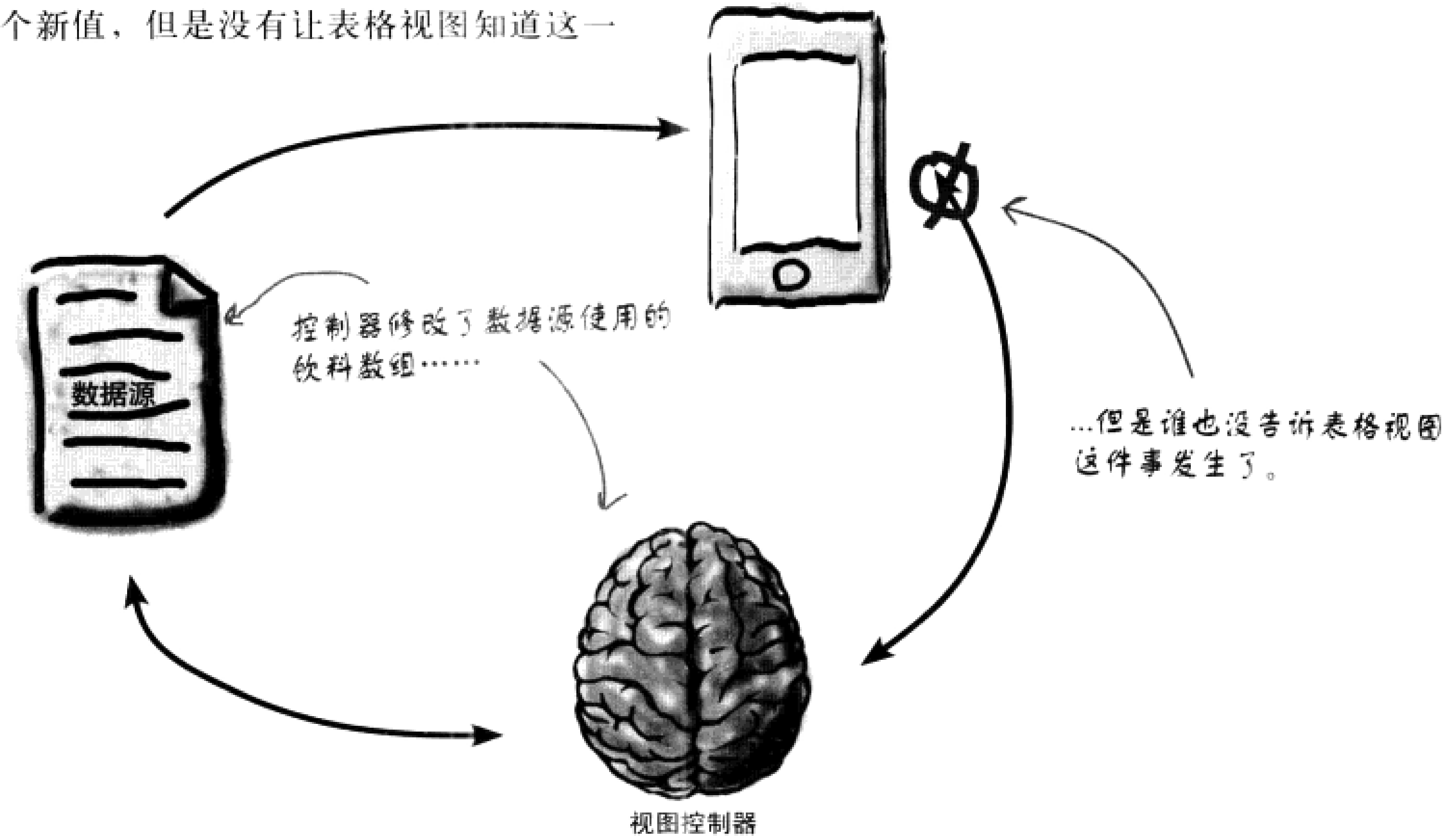


问题在于，名单被更改了，却没有人告诉她。其实仅仅需要一句简单的“注意”来提醒妮柯，让她知道名单有变化，那样餐厅的新投资人就不会消失了……还有她的工钱。

【拼吾爱-最新编程资源的分享下载站】

表格视图不知道数据发生变化

表格视图为了尽可能地提高性能，做了许多工作。结果就是，如果你只在数据源中改变了值而不告诉它，它是不会知道发生了什么事情的。对我们来说，我们在数据源使用的数组里添加了一个新值，但是没有让表格视图知道这一点。

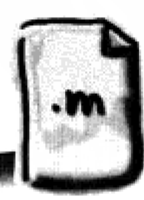


你得请求表格视图重新载入数据

由于我们修改了数据源使用的底层数据，所以刷新表格最简单的办法就是让它重新载入数据。向它发送reloadData消息就可以做到这一点。这样会告诉表格视图重新构建所有内容——有多少段落，这些段落的段头段尾，以及数据行，等等。

```
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    [self.tableView reloadData];
}
```

在RootViewController出现之前，请求表格视图重新载入数据。



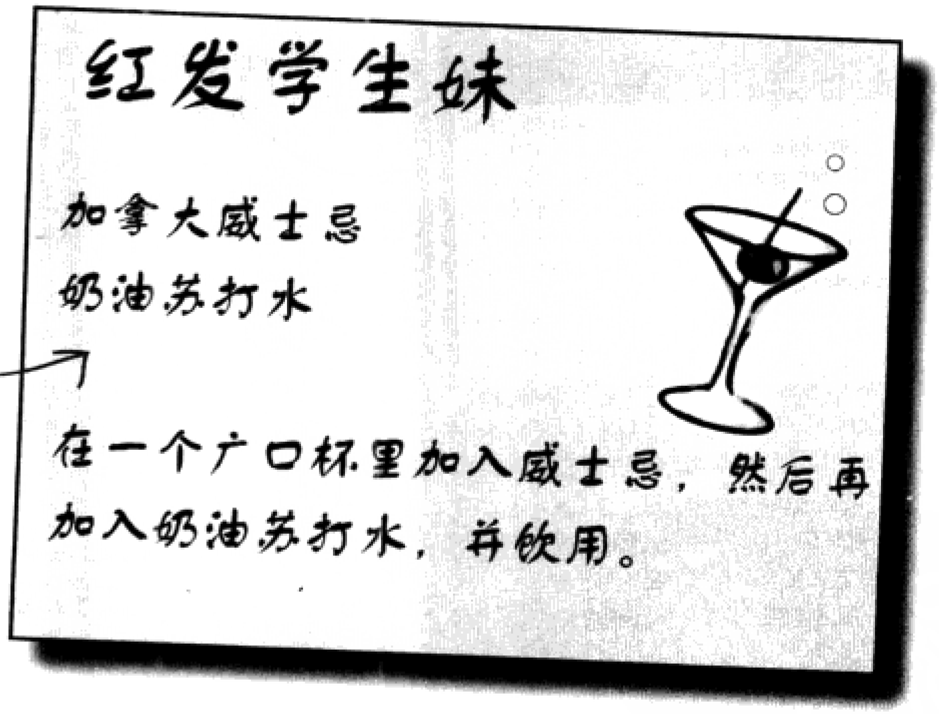
RootViewController.m

【拼吾爱-最新编程资源的分享下载站】

运行测试

更新你的RootViewController.m，让表格视图在显示之前，刷新自己的数据。然后再试一试添加一个新饮料。

把这个加到你的应用中。



【拼吾爱-最新编程资源的分享下载站】

运行测试

为了正确地测试这个应用，请点击加号按钮并在细节视图中输入新饮料的数据。做完以后，点击save进行保存。

现在回到列表视图之后，发生了什么？



【拼吾爱-最新编程资源的分享下载站】

有问必答

问：让表格重新载入全部数据看起来动作太大了。我真的应该这样做吗？

答： 这是刷新表格数据最简单的方法，但并不是最高效的。这取决于你怎样使用表格。如果在表格可见的情况下对其进行修改，你可以调用beginUpdates和endUpdates来告诉它你要进行一系列的修改，它就会帮你将这些修改用动画表示出来，这样你就不必调用reloadData了。也有其他一些方法，可以只重新加载指定段落的特定行。你选用哪一种取决于你的应用程序、对数据中的变化了解到何种程度，以及数据集的大小。

问：我们没有给取消按钮添加任何代码。在那儿我们不要做些什么吗？

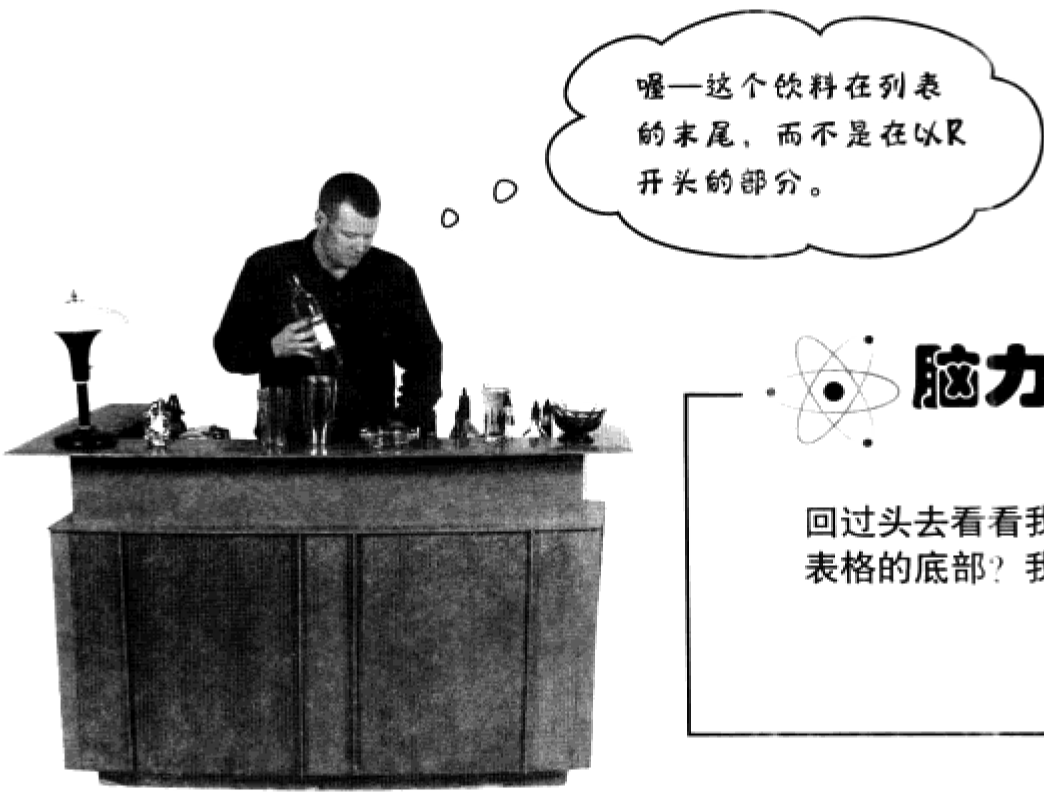
答： 不用——取消按钮就是用来让AddDrinkViewController消失的。这就会清除任何与控制器有关的内存，并丢弃任何用户在字段中输入的数据。只要我们没有对饮料数组进行操作，就能正确地取消用户启动的任何动作。

问：为什么我在调试器中打开饮料数组和字典时，无法看到饮料信息？

答： 这就是使用一个通用类，比如NSMutableDictionary，来存储饮料的不利之一。调试器知道这个类是字典，但是也仅此而已，因为所有的键和值都是动态的。你可以在调试控制台中获得这些内容，但不如我们在调试中查看类的真正属性那么方便。

问：我们真的需要用调试器来找出问题吗？我不能仅仅用NSLog将数组中的条目数打印出来吗？

答： 当然，但是那样你就不能再练习一次调试了 ... :-)



脑力练习

回过头去看看我们的调试工作。为什么饮料出现在表格的底部？我们需要怎么做？

【拼吾爱-最新编程资源的分享下载站】

数组的顺序也乱了

我们的表格视图直接从饮料数组中得到信息。实际上，我们只不过是在cellForRowAtIndexPath:中把行号映射到数组中的下标而已。

RootViewController.m

```
        // Configure the cell.
        cell.textLabel.text = [[self.drinks objectAtIndex:indexPath.row]
valueForKey:NAME_KEY];
```

我们将行号直接映射到数组中的下标。所以41行就是数组中的第41个点——也就是我们的新饮料。

我们可以用NSSortDescriptor来对数组排序

为了让表格视图能够正确排序，我们需要对数据数组进行排序。NSSortDescriptors正好可以做到这一点。你可以通过指定一个属性来告诉描述符要比较什么，用一个可选的选择子说明如何进行比较，还有就是按照何种顺序显示信息。对于我们来说，希望用饮料的名字按照字母顺序排序。

我们希望NSSortDescriptor根据饮料名称进行排序。

```
    // 因为我们刚刚添加了新饮料，所以要对数组进行排序
    NSSortDescriptor *nameSorter = [[NSSortDescriptor alloc]
initWithKey:NAME_KEY ascending:YES selector:@selector(caseInsensitiveCompare
:)];
    [drinkArray sortUsingDescriptors:[NSArray arrayWithObject:nameSorter]];
    [nameSorter release];
```

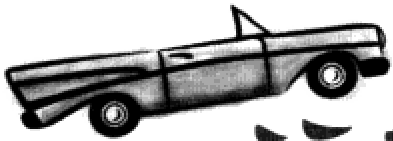
要进行排序很简单，让数组用我们的描述符对自身进行排序就可以了。

如果我们不提供选择子，它会进行区别大小写的比较，但是我们不要区别大小写。

将这段代码加入到保存方法中，放在将数据加到数组之后，但是位于视图从堆栈中弹出之前。

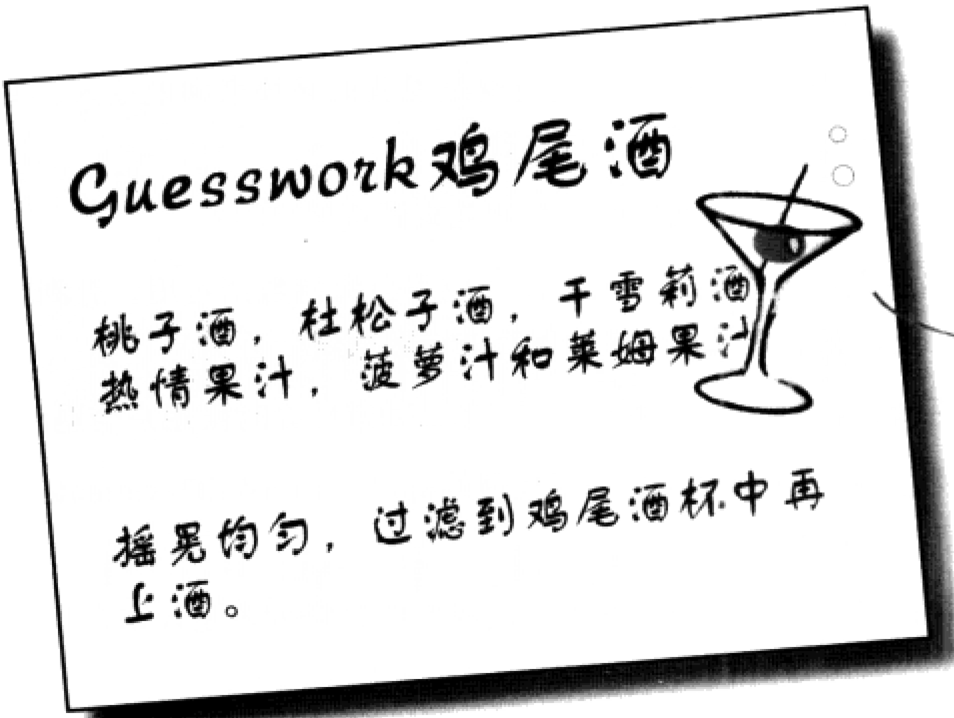
RootViewController.m

【拼吾爱-最新编程资源的分享下载站】



运行测试

将排序代码加到AddDrinkViewController中，然后运行应用。让我们再添加一个饮料：这次它就会出现在正确的位置上了。





太棒了，这个新饮料就位了，但是以前输入的红发学生妹这种饮料呢？我们是不是得处理一下，让保存可以更持久一些？



我们的数据一退出就全都丢失了……

我们很乐观地将新饮料加入了数组中，但是显然新的数组没法存活过应用退出与重启的过程。

我们该做些什么？什么时候行动？

【拼吾爱-最新编程资源的分享下载站】



吉姆：好吧，这么说我们应该在每个新饮料添加进来以后就保存数组，对吗？这样就可以保证我们的数据是正确的。

弗兰克：没那么容易。你得考虑速度、内存管理这整套东西。

乔：这有什么问题？不过是个很小的数组。

弗兰克：但这意味着每次你添加饮料都要保存数据。

吉姆：噢，我明白了，这也就是说我们必须来回地读入、写回数组很多次。这看起来很浪费。

乔：好了，那么我们准备在哪儿来做这件事呢？

弗兰克：当我们退出的时候！应用会保持数据可用，直到关闭为止，然后如果不想办法保存的话数据就会丢失。

吉姆：我们该怎么做呢？我们怎么知道什么时候用户会退出？

弗兰克：嗯……放在应用代理的applicationWillTerminate方法里怎么样？

乔：但是应用代理根本不知道我们的饮料列表是什么，也不知道往哪里存储它……

弗兰克：说的对。据说UIApplicationDelegate也有一个通知。我打赌我们可以用这个……

有问必答

问： 哪种通知可以告诉我们应用程序要退出呢？

答： iPhone OS会在应用退出前发送UIApplicationWillTerminateNotification。

问： 我需要去注册接收它吗？

答： 是——正如任何其他通知一样。

问： 如果用户按下主按钮或者电话铃响了……会怎样？

答： 只要你的应用程序是正常退出的，不管是通过代码还是用户按下按钮，或者其他什么事触发了iPhone切换应用程序（比如电话打进来而用户又决定接听），你都会进入applicationWillTerminate方法。实际上只有一种情况才不会进入……

问： 如果应用崩溃了会怎样？

答： 那你就不会收到通知。这种情况下数据会丢失。你需要在确保不丢失数据的重要性和频繁保存数据对性能的影响之间做出权衡。对于我们来说，只要在退出时保存就可以了。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来

依靠你使用API的技巧以及吉姆、弗兰克和乔的讨论结果，想办法实现对数组进行存储。更新RootViewController.m和RootViewController.h文件，处理存储工作。

1 **添加代码，保存新的字典plist。**

实现一个方法，在收到UIApplicationWillTerminateNotification的时候调用。我们打算给一小段代码供你使用。这段代码只能在模拟器上工作，不过我们会在第7章中对其重新修订。

```
NSString *path = [[NSBundle mainBundle]
      pathForResource:@"DrinkDirections" ofType:@"plist"];
[self.drinks writeToFile:path atomically:YES];
```

2 **注册订阅UIApplicationWillTerminateNotification通知。**

我们知道AppDelegate中的applicationWillTerminate:方法会在应用程序关闭时被调用，但是实际上是RootViewController拥有所有数据。让RootViewController像AddDrinkViewController一样，注册订阅UIApplicationWillTerminateNotification，不同之处在于，要把注册和取消注册代码分别添加到viewDidLoad和viewDidUnload中。



留神!

这段代码只能用在模拟器上！

这段用来保存plist的代码在模拟器上工作状况良好，但是在真实设备上就悲惨地失败了。问题在于文件权限以及应用允许保存数据的位置。我们会在第7章中进一步讲解这个问题，现在先用着这个版本。这是一个很好的例子，展示了能够运行在模拟器上，却在真实设备上行为有差异的情况。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

依靠你使用API的技巧以及吉姆、弗兰克和乔的讨论结果，想办法实现对数组进行存储。更新RootViewController.m和RootViewController.h文件，处理存储工作。

将这些代码加到viewDidLoad中。

RootViewController.m

```
// 注册应用程序退出的通知，这样我们就可以保存数据了
[[NSNotificationCenter defaultCenter] addObserver:self
      selector:@selector(applicationWillTerminate:)
      name:UIApplicationWillTerminateNotification object:nil];
```

也别忘了在RootViewController.h中声明这个方法。

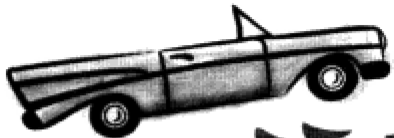
```
- (void)applicationWillTerminate:(NSNotification *)notification {
    NSString *path = [[NSBundle mainBundle]
          pathForResource:@"DrinkDirections" ofType:@"plist"];
    [self.drinks writeToFile:path atomically:YES];
}
```

这就是在真实设备上会出问题的代码。我们在下一章里会再次用到这段代码（并改正它）——现在先凑合用吧....

把这个加到viewDidUnload中。

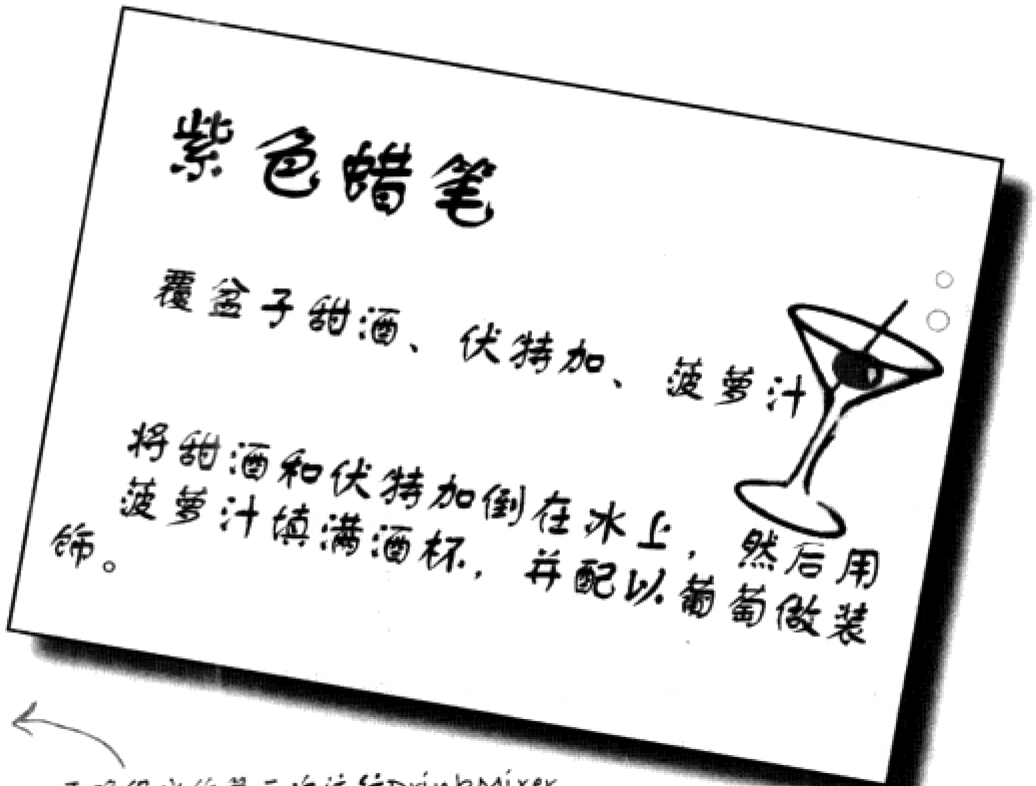
```
// 取消对通知的注册订阅
[[NSNotificationCenter defaultCenter] removeObserver:self];
```

【拼吾爱-最新编程资源的分享下载站】



运行测试





要确保当你第二次运行DrinkMixer的时候，点击的是模拟器里的图标。不要去按Build and Debug！

作者注：我们曾经想显示两次同样的屏幕截图，但是觉得那样也无法证明数据被保存下来了，并且是在按下主按钮之后，再次回到程序中显示的。



留神！

在Xcode中停止应用并再次“Build and Run”，与按下主按钮然后在模拟器中再次启动应用，是不一样的！

当你用Xcode的停止按钮停下程序时，是将应用就地杀死了。不会发送终止通知，所以也不会进行保存——应用就是停下来了。与之类似，当你点击“Build and Debug”按钮时，Xcode会将应用程序重新安装到你的设备上然后再启动。要想测试我们的加载和保存代码，一定要点击模拟器里面的图标来重新启动应用。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：既然数组知道该如何保存自身……我随便放任何对象进去都能保存到plist中去吗？

答： 不行——不是任何对象都可以。数组是使用一种名为NSCoding的Cocoa技术来进行保存和加载的。你要保存和加载的对象，都必须遵循NSCoding协议，协议中包括initWithCoder和encodeWithCoder两个方法——分别进行加载和保存。你得遵循NSCoding协议并提供这些方法，以便对数组进行序列化输入输出。不过NSDictionary确实遵循NSCoding（里面的字符串也一样），这就是我们可以如此轻松地进行保存和加载的原因。

问：给我们一些在真实设备上无法使用的代码有什么用处？这些代码到底有什么问题？

答： 哎呀，为了探询究竟，你最好在真实设备上运行这段代码。然后好好想想为什么它和你期望的运行结果不一样。在下一章里我们会好好讲一讲这个问题。给你一点提示，问题和我们保存数据的位置有关。这也是一个现实世界的例子，让你能见到有些东西只能用在模拟器上，在真实设备上效果就不一样了。所以你总是要在两种环境下都做测试。

问：我们能不能不去注册那个退出通知，而是修改AppDelegate，让它可以从RootViewController那里得到饮料数组，并在代理里保存它吗？

答： 是，你这么做也可以。这主要是一个设计风格上的考虑，而不是其他什么必须的守则。目前AppDelegate根本不了解任何关于我们的plist、饮料数组甚至是RootViewController的事情（否则就要令其对外可见）。你可能有不同意见，但如果我们将对每个视图进行保存加载所需要了解的内容都暴露给AppDelegate，就会对封装造成破坏。由于我们仅仅需要保存一个数组，所以不管怎么做都不会有太大的问题，但是如果你有许多视图都需要保存信息或者有复杂的持久化代码，通常来说还是让需要这些信息的类来处理它更清晰一些，而不是把它们都塞给AppDelegate。不过从技术角度来说，两种办法都能完成任务。

问：为什么我们要在viewDidLoad和viewDidUnload方法中去注册和取消注册，而不是在*Appear方法中？

答： 问题在于这些方法被调用的时机和频繁程度。每次视图要显示的时候，都会调用viewWillAppear。这也还可以——我们会在表格视图显示之前得到调用，可以注册通知。然而viewWillDisappear会在马上要显示细节或者添加饮料视图之前被调用（因为RootViewController要被隐藏起来了）。如果我们在那里取消对通知的注册，而用户在查看饮料细节的时候决定退出，我们就没法得到终止通知了。

例如，如果用户添加了一个新饮料，回到RootViewController，然后点击他刚刚添加的饮料，好看看输入是否全都正确。我们显示出细节视图，用户很高兴，然后退出了应用。而我们的RootViewController已经取消了对终止通知的注册，饮料就丢了。与之相反，我们使用了load和unload方法，这两个方法是在视图从nib中加载和卸载时调用的。因为视图在应用程序中始终被使用着，它们只会在应用启动和关闭的时候才会被调用。

问：第二次启动DrinkMixer的时候，按“Build and Run”按钮和点击图标到底有什么不同？

答： 这是由于我们保存数据的方式造成的。我们会在下一章里进一步讲这个问题，不过简单来说，就是当你按下“Build and Debug”的时候，Xcode会编译应用程序并将其安装到模拟器上。这样就会把修改过的饮料plist替换成我们随应用程序自带的那个，这样自己添加的饮料就丢失了。对于这种后果大家都感到非常非常遗憾。

【拼吾爱-最新编程资源的分享下载站】



1. 删除不再使用的饮料，保持列表小巧易用。

2. 编辑已经存在于列表中的饮料的成份。

脑力练习

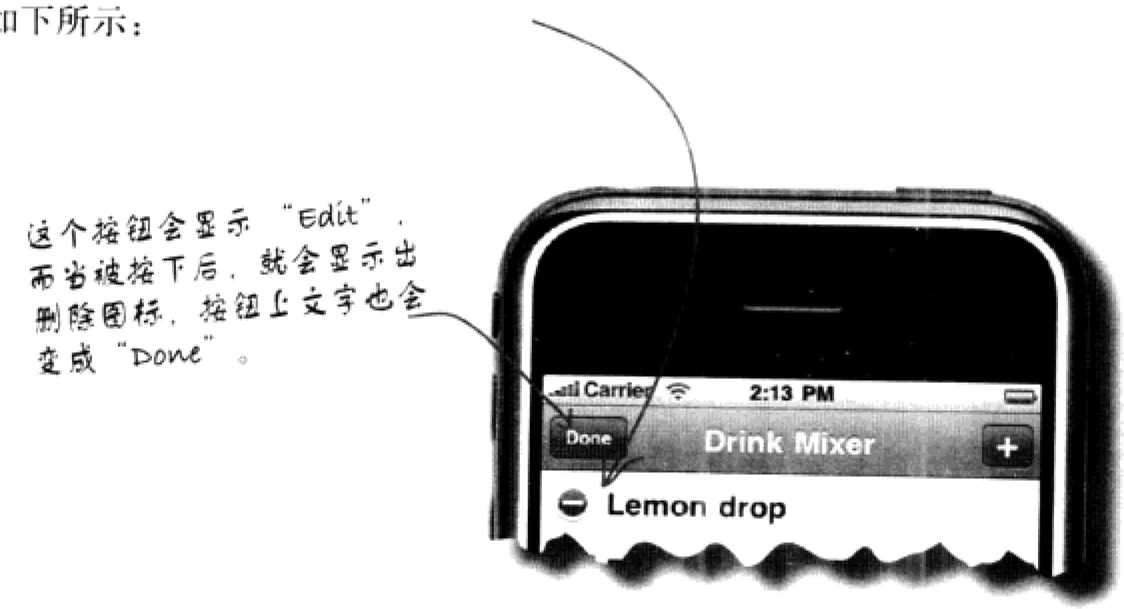
我们怎样才能实现这些东西呢？我们需要在应用中的什么地方来处理这些问题？

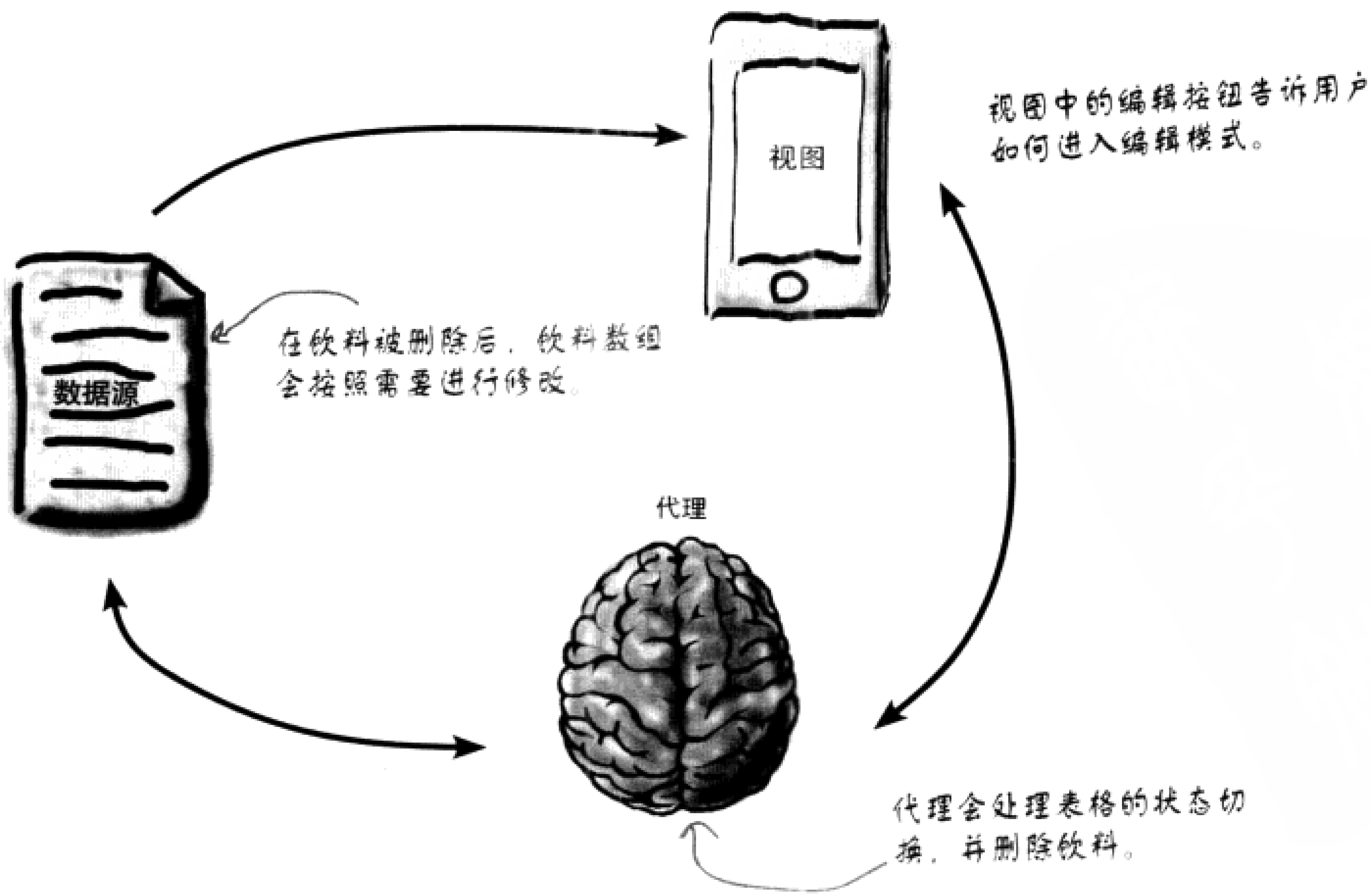
【拼吾爱-最新编程资源的分享下载站】

表格视图内建支持编辑与删除

好消息！表格视图自己就几乎完全支持我们需要的删除数据功能。这种行为与实现保存和取消按钮有一点相似，而且大部分都是预先加载的功能。

处于编辑模式时，主视图中的导航控件上面会增加一个编辑按钮，当这个按钮被按下时，那些能够被选中并删除的表格单元的左侧会出现一个指示器，如下所示：





【拼吾爱-最新编程资源的分享下载站】



打造编辑视图

使用下面的视图，写出编辑视图中每一部分应该做的事。



【拼吾爱-最新编程资源的分享下载站】

打造编辑视图答案

使用下面的视图，写出编辑视图中每一部分应该做的事。



【拼吾爱-最新编程资源的分享下载站】



练习

Xcode模板自带了不少我们需要的代码，而且目前你也已经对表格视图和RootViewController相当熟悉了。所以对于下面要实现的内容，我们会给你一点提示，但从此以后就要靠你自己来完成了。

❶ 在根视图中添加编辑按钮。

我们需要在导航栏的左上角放一个编辑按钮。UITableViewController的模板代码自带了我们需要的所有内容；只要把viewDidLoad里面的代码行去掉注释就可以了。

❷ 实现tableView:commitEditingStyle:forRowAtIndexPath。

表格视图进入编辑模式后，一旦用户试图删除一行内容时，我们就会收到对这个方法的调用，删除的方法可以是用手指扫过表格行，或者点击删除指示器。这个方法的大部分内容也已经在模板代码里面了，不过你得加上根据变化更新数据源的代码。记住，我们已经将表格的行与数组中的下标对应起来了。最后一点，在这个变化之后你不必调用reloadData，因为我们会显式地请求表格视图删除该行。

❸ 更新didSelectRowAtIndexPath方法，以添加饮料。

对于编辑已有饮料这项功能所需的工作，我们的AddDrinkViewController几乎已经万事俱备了。更新一下didSelectRowAtIndexPath方法，如果在编辑模式的话，调用AddDrinkViewController，而不是DrinkDetailViewController。

❹ 确认Interface Builder知道表格是可编辑的。

检查一下饮料表格视图的“Allow Selection While Editing”选项，确保已经选中。

❺ 为我们的AddDrinkViewController增加编辑饮料的功能。

你需要告诉应用，现在需要编辑一个饮料的内容，而不是创建新的饮料，然后用现有的饮料信息填充各个控件，最后在保存时更新饮料的内容。

【拼吾爱-最新编程资源的分享下载站】

练习答案

Xcode模板自带了不少我们需要的代码，而且目前你也已经对表格视图和RootViewController相当熟悉了。所以对于下面要实现的内容，我们会给你一点提示，但从此以后就要靠你自己来完成了。

1 **在根视图中添加编辑按钮。**

我们需要在导航栏的左上角放一个编辑按钮。UITableViewController的模板代码自带了我们需要的所有内容；只要把viewDidLoad里面的代码行去掉注释就可以了。

在viewDidLoad中

```
// Uncomment the following line to display an Edit button in the navigation bar
for this view controller.
self.navigationItem.leftBarButtonItem = self.editButtonItem;
```

UITableViewController自带了对编辑按钮的支持。我们要做的就是把它添加到导航栏中。

RootViewController.m

2 **实现tableView:commitEditingStyle:forRowAtIndexPath。**

表格视图进入编辑模式后，一旦用户试图删除一行内容时，我们就会收到对这个方法的调用，删除的方法可以是用手指扫过表格行，或者点击删除指示器。这个方法的大部分内容也已经在模板代码里面了，不过你得加上根据变化更新数据源的代码。记住，我们已经将表格的行与数组中的下标对应起来了。最后一点，在这个变化之后你不必调用reloadData，因为我们会显式地请求表格视图删除该行。

```
// Override to support editing the table view.
- (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEdit
ingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath {
  if (editingStyle == UITableViewCellEditingStyleDelete) {
    // Delete the row from the data source.
    [self.drinks removeObjectAtIndex:indexPath.row];
    [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
      withRowAnimation:UITableViewRowAnimationFade];
  }
  else if (editingStyle == UITableViewCellEditingStyleInsert) {
  }
}
```

用removeObjectAtIndex来清理数据源。

RootViewController.m

【拼吾爱-最新编程资源的分享下载站】

3 更新didSelectRowAtIndexPath方法，以添加饮料。

对于编辑已有饮料这项功能所需的工作，我们的AddDrinkViewController几乎已经万事俱备了。更新一下didSelectRowAtIndexPath方法，如果在编辑模式的话，调用AddDrinkViewController，而不是DrinkDetailViewController。

```
// Override to support row selection in the table view.
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)
indexPath {
  if (!self.editing) {
    DrinkDetailViewController *drinkDetailViewController =
[[DrinkDetailViewController alloc] initWithNibName:@"DrinkDetailViewController"
bundle:nil];
    drinkDetailViewController.drink = [self.drinks objectAtIndex:indexPath.row];
    [self.navigationController pushViewController:drinkDetailViewController
animated:YES];
    [drinkDetailViewController release];
  }
  else {
    AddDrinkViewController *editingDrinkVC = [[AddDrinkViewController
alloc] initWithNibName:@"DrinkDetailViewController" bundle:nil];
    UINavigationController *editingNavCon = [[UINavigationController alloc]
initWithRootViewController:editingDrinkVC];
    editingDrinkVC.drink = [self.drinks objectAtIndex:indexPath.row];
    editingDrinkVC.drinkArray = self.drinks;
    [self.navigationController presentModalViewController:editingNavCon
animated:YES];
    [editingDrinkVC release];
    [editingNavCon release];
  }
}
```

首先我们得检查是否处于编辑模式。如果不是，只需显示正常的细节视图。

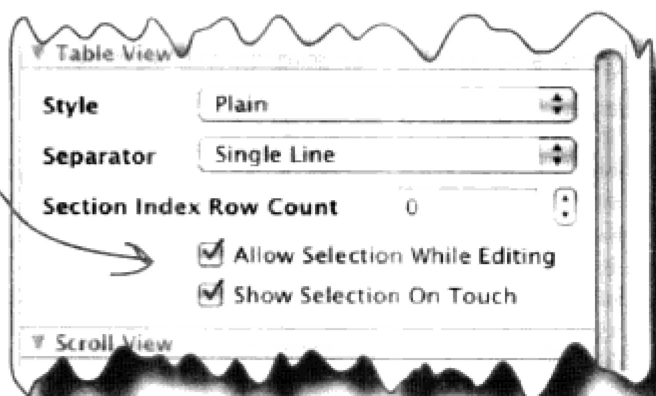
如果处于编辑模式，创建一个AddDrinkViewController，并设置要编辑的饮料以及饮料数组。我们马上就会修正AddDrinkViewController…



RootViewController.m

4 确认Interface Builder知道表格是可编辑的。

检查一下饮料表格视图的“Allow Selection While Editing”选项，确保已经选中。



只剩下AddDrinkViewController了……

【拼吾爱-最新编程资源的分享下载站】



Xcode模板自带了不少我们需要的代码，而且目前你也已经对表格视图和RootViewController相当熟悉了。所以对于下面要实现的内容，我们会给你一点提示，但从此以后就要靠你自己来完成了。

5 **为我们的AddDrinkViewController增加编辑饮料的功能。**

你需要告诉应用，现在需要编辑一个饮料的内容，而不是创建新的饮料，然后用现有的饮料信息填充各个控件，最后在保存时更新饮料的内容。

```
- (void)viewWillAppear: (BOOL)animated {
        [super viewWillAppear:animated];

        NSLog(@"Registering for keyboard events");
        [[NSNotificationCenter defaultCenter] addObserver:self
                selector:@selector(keyboardWillShow:)
                name:UIKeyboardWillShowNotification object:self.view.window];

        [[NSNotificationCenter defaultCenter] addObserver:self
                selector:@selector(keyboardWillHide:)
                name:UIKeyboardDidHideNotification object:nil];

        // 初始状态下键盘是隐藏的，所以将变量重置
        keyboardVisible = NO;

        if (self.drink != nil) {
          nameTextField.text = [self.drink objectForKey:NAME_KEY];
          ingredientsTextView.text = [self.drink objectForKey:INGREDIENTS_
KEY];
          directionsTextView.text = [self.drink objectForKey:DIRECTIONS_
KEY];
        }
}
```

如果我们设置了drink，就意味着我们是要编辑这个饮料，而不是创建一个新的。我们需要用当前饮料的信息填充各字段的内容。



AddDrinkViewController.m

【拼吾爱-最新编程资源的分享下载站】

如果设置了drink，那么需要对它进行更新。我们可以更新现有对象，也可以替换掉它。由于我们无论如何都要对整个数组重新排序(以防饮料名字变了)，所以就删除了老的对象，重新添加了一个。

```
- (IBAction) save: (id) sender {
    NSLog(@"Save pressed!");

    if (drink != nil) {
        // 我们在使用一个已有的饮料，所以将它从
        // 数组中删除，换成一个新的
        [drinkArray removeObject:drink];
        self.drink = nil; // 这样就会释放我们的引用
    }

    // 现在用新的值创建一个新的饮料字典
    NSMutableDictionary* newDrink = [[NSMutableDictionary alloc] init];
    [newDrink setValue:nameTextField.text forKey:NAME_KEY];
    [newDrink setValue:ingredientsTextView.text forKey:INGREDIENTS_KEY];
    [newDrink setValue:directionsTextView.text forKey:DIRECTIONS_KEY];

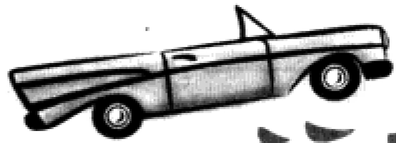
    // 将其添加到主饮料数组中，并释放对它的引用
    [drinkArray addObject:newDrink];
    [newDrink release];

    // 然后进行排序，因为现有饮料的名称有可能发生变化，
    // 也可能是一种全新的饮料。
  NSSortDescriptor *nameSorter = [[NSSortDescriptor alloc] initWithKey:NAME_KEY
ascending:YES selector:@selector(caseInsensitiveCompare:)];
    [drinkArray sortUsingDescriptors:[NSArray arrayWithObject:nameSorter]];
  [nameSorter release];

    // 然后将细节视图从栈中弹出
    [self.navigationController dismissModalViewControllerAnimated:YES];
}
```

AddDrinkViewController.m

【拼吾爱-最新编程资源的分享下载站】



运行测试

在你的应用中加入编辑功能所需的修改，并尝试一下它的功能。你应该能够删除饮料，也可以对其进行调整，完全随心所欲。不过要记得重启应用的时候要点击图标；否则所做的修改就会丢失。





将你的应用重新提交到AppStore，这次……

【拼吾爱-最新编程资源的分享下载站】

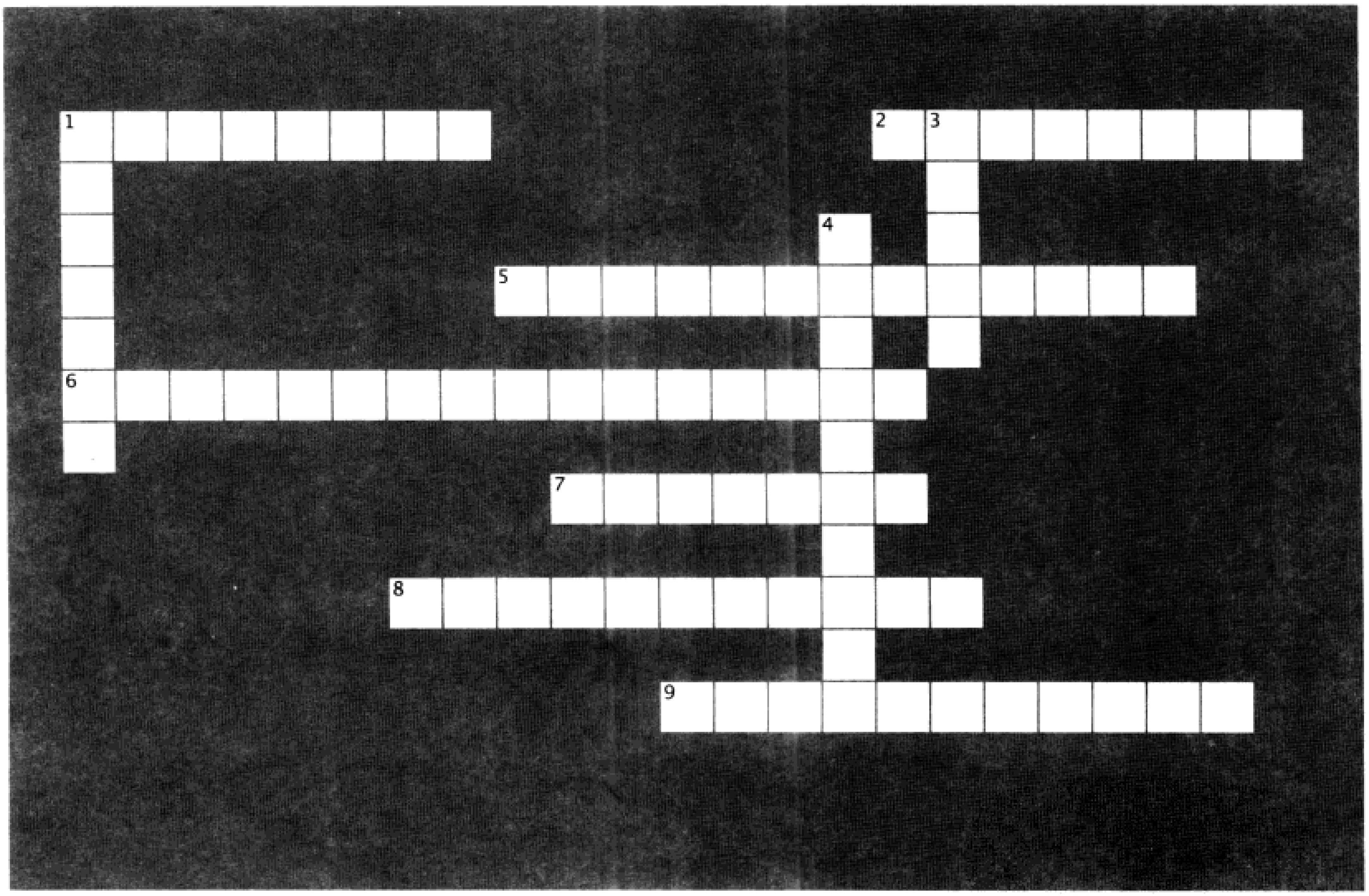


DrinkMixer排名第一！热烈祝贺！

【拼吾爱-最新编程资源的分享下载站】

导航控制器纵横字谜

让我们检查一下你对滚动视图、导航控制器和表格视图方面词汇的掌握情况！



横向提示

1. 一个字段如果可以对其进行修改，则是 ________ 的。
2. 数组用 ________ 进行保存和加载。
5. 可以在系统级别传递的事件叫做 ________。
6. 对数据进行排序需要使用 ________。
7. 所有系统事件都要通过________中心发送。
8. 如果不设置 ________，滚动视图就无法正常工作。
9. viewWillAppear和________的调用时机不同。

纵向提示

1. 表格视图内建支持________ 。
3. 键盘事件会告诉你键盘的________和尺寸。
4. ________会处理滚动条、移动、缩放以及视图中显示的内容。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：我喜欢表格视图的自动编辑功能，不过我怎么才能做到像iPhone上编辑联系人的时候出现的“添加新地址”行那么酷啊？

答： 比你想像的要简单得多。基本上，在表格视图处于编辑模式时，告诉它的行数要比数据中实际有的多一行。然后，在cellForRowAtIndexPath方法中，检查表格视图请求的行是否超过数据末尾一行。如果是，就返回一个单元格，里面写着“添加新地址”或者随便什么内容。最后，在你的didSelectRowAtIndexPath方法中，检查选中的行是否比你的数据多一行，如果是，你就知道选中的是这一行。

问：我们还没讲到如何移动数据行呢，但是我见到过有表格这样做。这很难吗？

答： 不难，表格视图这部分很容易；不过数据源这边就要麻烦一些。如果你支持对行进行移动，只要实现tableview:moveRowAtIndexPath:toIndexPath就可以了(表格视图在允许用户重新布置单元格之前，会检查你是否提供了这个方法)。用户会在处于编辑模式的单元格旁边看到一个行的把手。当用户移动行时，你的新方法就会被调用，而行开始所在的位置和新位置都会作为IndexPath传给这个方法。你的任务就是更新数据源，确保数据也按照相同的方式移动。你也可以实现tableview:canMoveRowAtIndexPath让用户只能移动某些特定行。如果你感兴趣，代理中甚至还有一些更精细的控制方式，比如阻止用户将单元格移动到某个特定段落中。

问：如果我不想让用户删除某一行该怎么办？还能支持对其他一些行进行编辑吗？

答： 绝对没问题。只要实现tableview:canEditRowAtIndexPath:并对你不想让编辑的行返回NO就行了。

问：当我们编辑饮料时，将数组中的对象替换掉了。如果还有其他一些视图，对原来的对象有引用怎么办？

答： 问得太好了。简单的答案就是，不管你想什么办法，都会有麻烦的。如果其他某些视图对我们删除的对象有引用的话，倒不会出现太大的问题，因为持有计数应该至少还有1；我们删除的对象不会被销毁。然而，其他的视图就无法见到用户所做的任何修改了，因为我们把这些修改放在一个新的字典里面了。即使它们还有旧的字典，也无法得知值已经发生了变化。有几种办法可以处理这种情况。一个选择是你可以修改代码，将原来的对象留在数组中，只是对它的内容进行修改，然后确保其他视图会在viewWillAppear或者其他什么类似的地方对自己进行刷新。另一个选择是你可以发送一个自定义的通知，告知饮料数组发生了变化，或者某个特定饮料被修改了。对此感兴趣的视图可以订阅这个通知。

问：我们不是要注意效率问题吗？删除一个饮料再重新添加一个是不是有点效率低下啊？

答： 这种方法不是最高效的，因为需要在数组中查找对象，并在重新插入前首先删除对象，但是为了代码清晰起见，我们决定还是用这种比较简单的方法来做示范。然而，无论我们采取哪一种方式，都必须得重新对数组进行排序，因为饮料的名称可能会在编辑过程中被修改（它的字母顺序的位置也随之变化）。

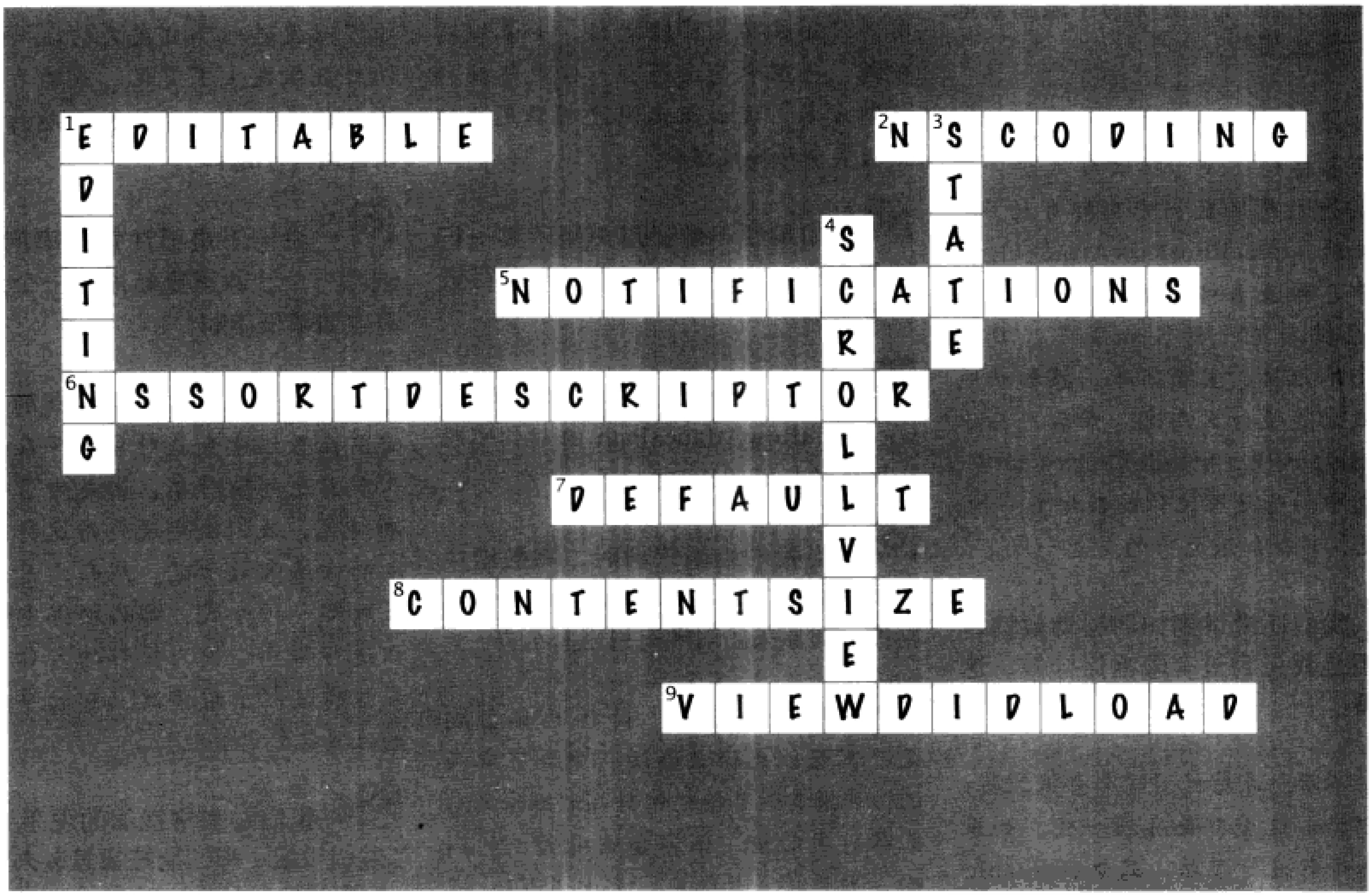
问：我们在表格视图的左手边添加了编辑按钮，那回退按钮怎么办？这儿一般不是它呆的地方吗？

答： 确实如此。当你要同时用到添加按钮、编辑按钮以及回退按钮的时候，就会遇到地盘不够的问题。我们的解决方法还是不错的，不过如果以后你遇到导航控制器开始变得拥挤的情况，一定要确认应用的工作流程还仍然与你的意图相同。

【拼吾爱-最新编程资源的分享下载站】

导航控制器纵横字谜答案

让我们检查一下你对滚动视图、导航控制器和表格视图方面词汇的掌握情况！



横向提示

1. 一个字段如果可以对其进行修改，则是 ________ 的。[EDITABLE]
2. 数组用 ________ 进行保存和加载。[NSCODING]
5. 可以在系统级别传递的事件叫做 ________。[NOTIFICATIONS]
6. 对数据进行排序需要使用 ________。[NSSORTDESCRIPTOR]
7. 所有系统事件都要通过 ________ 中心发送。[DEFAULT]
8. 如果不设置 ________，滚动视图就无法正常工作。[CONTENTSIZE]
9. viewWillAppear和 ________ 的调用时机不同。[VIEWDIDLOAD]

纵向提示

1. 表格视图内建支持 ________。[EDITING]
3. 键盘事件会告诉你键盘的 ________ 和尺寸。[STATE]
4. ________ 会处理滚动条、移动、缩放以及视图中显示的内容。[SCROLLVIEW]

【拼吾爱-最新编程资源的分享下载站】



你的iPhone开发工具箱

你已经消化吸收了第6章，在你的工具箱中已经添上了对数据进行保存、编辑以及排序。如果你想得到一份完整的书中功能提示的列表，请访问http://www.headfirstlabs.com/iphonedev。

滚动视图

像一个放大镜一样，只将视图中你需要的部分内容显示出来，剩下都滚动到屏幕之外。

要得到contentSize才能正常工作。

在Interface Builder中很容易构建。

通知

是系统级的事件，你可以在应用中对其进行监视和使用。

默认的通知中心会处理绝大部分通知。

不同的框架会使用不同的通知，你也可以创建自己的通知。

排序

可以用NSSortDescriptors对数组进行排序。

编辑表格视图

表格视图内建支持对内容进行编辑。

编辑按钮自带了许多功能，包括从表格视图中删除数据行的方法。

【拼吾爱-最新编程资源的分享下载站】

【拼吾爱-最新编程资源的分享下载站】

7 标签栏与Core Data

~~代理~~（或~~经纪人~~）企业应用



企业应用意味着以多种不同的方式来管理更多的数据。对于iPhone应用来说，各种大大小小的公司是一个重要市场。一个小型手持设备再配以专门定制的应用，对于那些雇员始终处于移动状态的公司来说，其用途是巨大的。这样的应用大多数都会管理大量数据，而从iPhone 3.x开始，系统就内置了对Core Data的支持。通过使用这个框架，并辅以另一个控制器——标签栏控制器，我们就能够打造一个为正义而战的应用！

【拼吾爱-最新编程资源的分享下载站】

HF赏金捕猎



鲍勃需要帮助。

赏金捕猎可不是文案工作；鲍勃抓捕逃犯的过程中需要许多信息。要想在工作过程中随身携带所有的案件资料，用他的iPhone是最理想不过了。下面是鲍勃对他的应用的要求：

1. 鲍勃需要一个逃犯列表。他必须了解每个人的状态，包括他正在追踪的，以及他已经捕获的。

2. 他想要一个能够快速显示的列表，里面仅包括已被捕获的逃犯。

3. 他还需要一幅画面来显示每个逃犯的详细信息，比如被通缉原因、最后出现的地点以及对他的悬赏是多少。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来

做一些设计工作的时间到了。你了解鲍勃的需求——请根据这些需求，勾画出你对这个应用的想法。

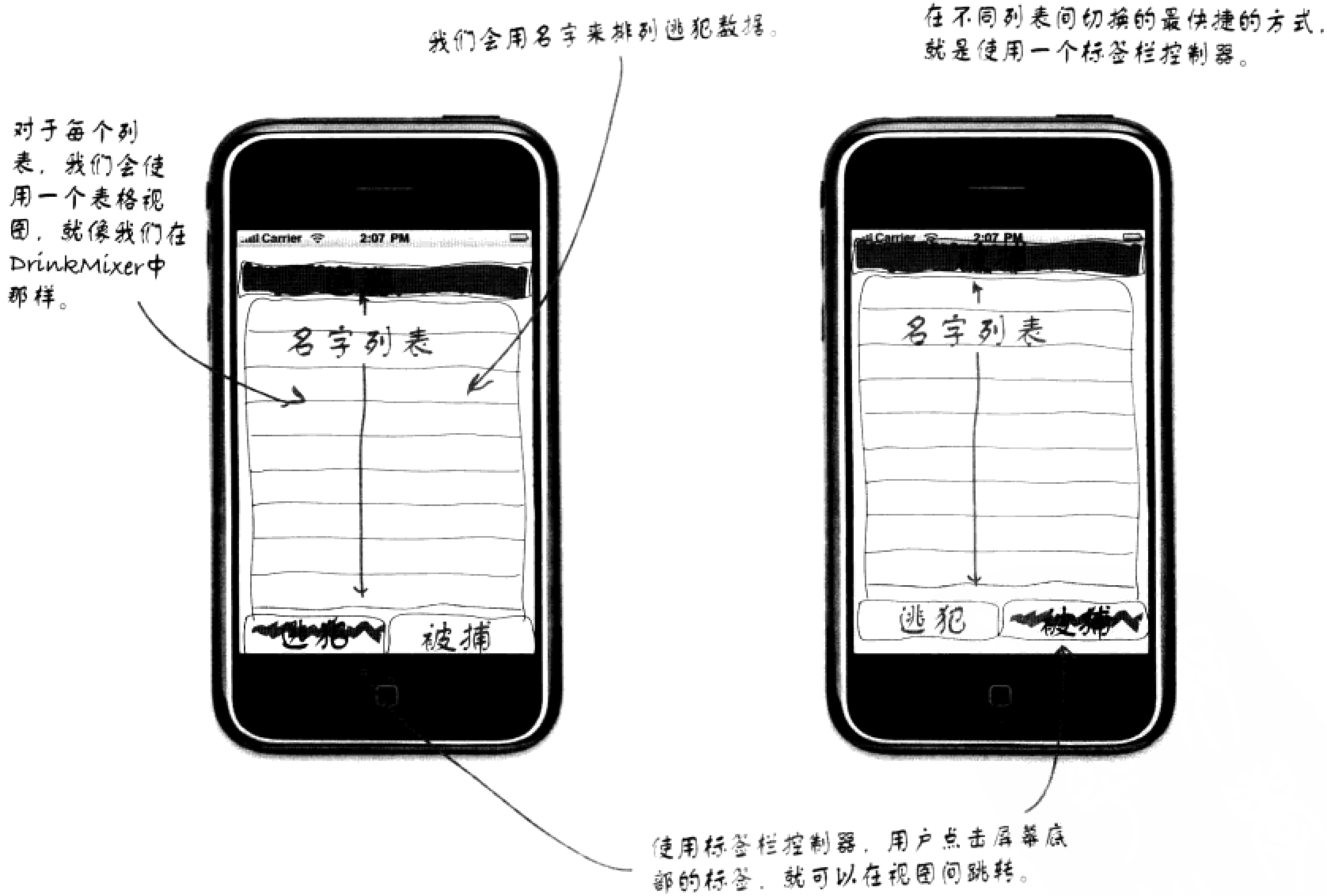
【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

我们大概会需要3个视图。利用鲍勃提供的内容，我们得出下面的成果。

1. 鲍勃需要一个逃犯列表。他要随时了解所有他正在追捕或者已经捕获的人的状态。

2. 他想要一个能够快速显示的列表，里面只显示已被捕的逃犯。



【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

3 鲍勃想要一幅画面来显示每个逃犯的详细信息。



为了管理数据，我们打算使用一种iPhone 3.x中引入的新技术——Core Data。这种技术可以为你的应用管理许多不同的数据类型。

点击逃犯的姓名，就可以看到每个逃犯的细节视图。



这一区域用来显示逃犯细节信息和备注。

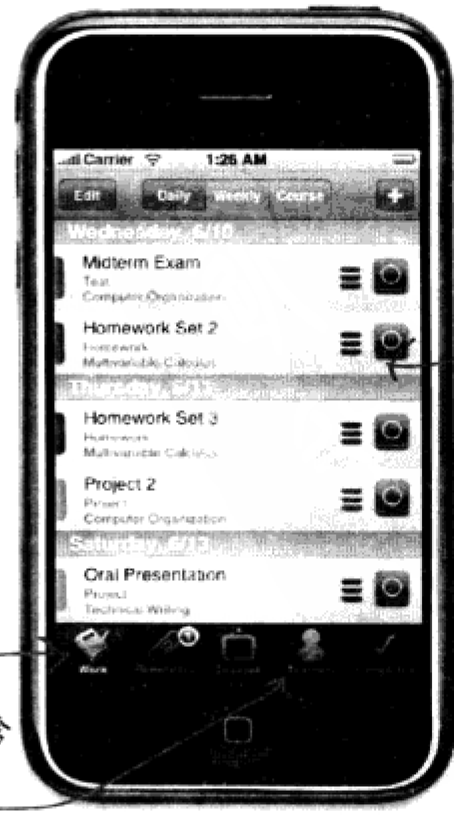
标签栏控制器仍然可见。

标签栏全接触

标签栏控制器是另一种常见的iPhone界面。与导航栏控制器不同，标签栏中没有视图堆栈。所有视图都是预先创建好的，每个标签和一个特定视图绑定，通过点击标签很容易就可以访问到视图。

标签栏更适合用于相互关联、却不一定有层次的任务或者数据。UITabBarController了解所有视图的状态，根据用户输入在它们之间切换。

用到标签栏的标准iPhone应用包括电话应用和iPod。



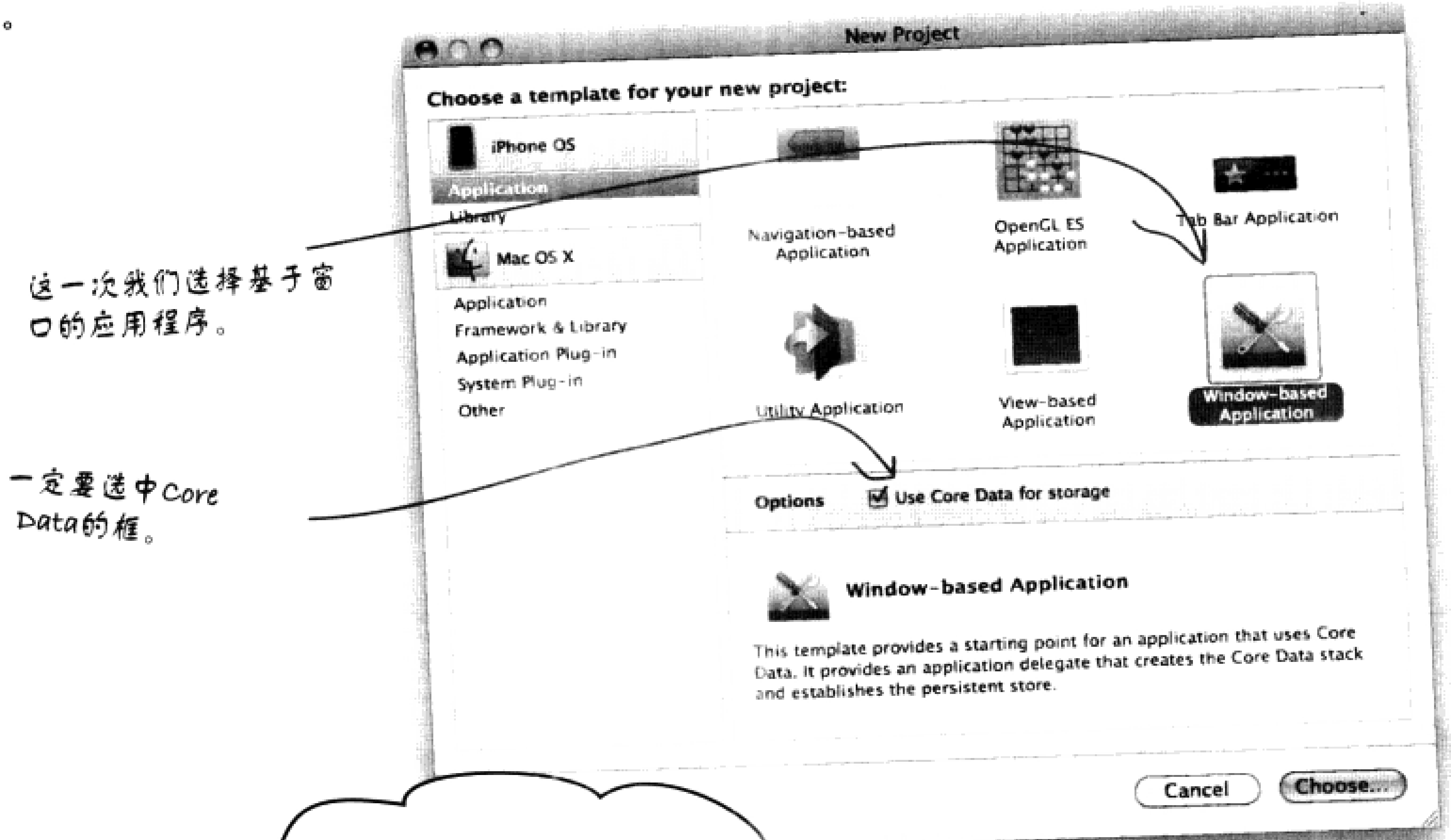
标签栏可以容纳任何你需要的视图。

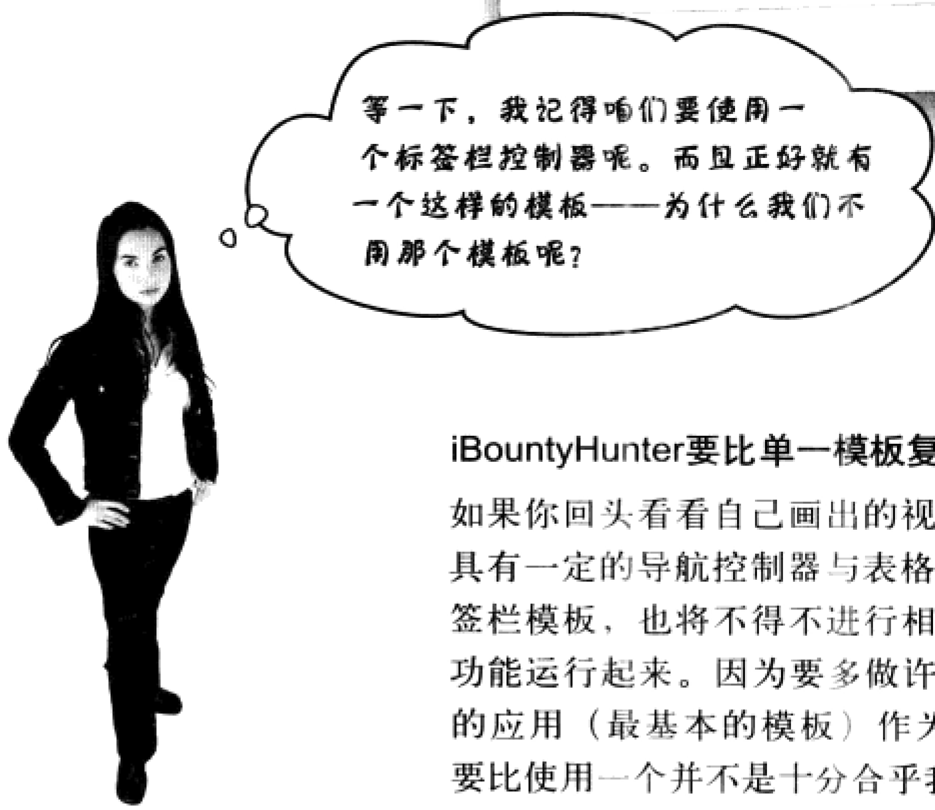
标签自身可以包含文字与图像。

【拼吾爱-最新编程资源的分享下载站】

为iBountyHunter选择一个模板

这一次，我们的应用要用到很多东西：导航控制器，标签栏，还有Core Data。Core Data对于许多模板来说都是一个可选的附加功能，这些模板中也包括基本的基于窗口的应用。我们将会用一个基于窗口的应用作为起点，并使用Interface Builder加上一点点代码，在其中加入标签栏以及导航控制器。





iBountyHunter要比单一模板复杂一些。

如果你回头看看自己画出的视图草图就会发现，我们还需要具有一定的导航控制器与表格视图的功能。即使我们用了标签栏模板，也将不得不进行相当多额外的工作，才能让这些功能运行起来。因为要多做许多工作，所以用一个基于窗口的应用（最基本的模板）作为起点，然后向里面添加功能，要比使用一个并不是十分合乎我们要求的模板更容易一些。

【拼吾爱-最新编程资源的分享下载站】



吉姆：好吧，现在我们该干什么呢？我们拥有的全部东西就只有一个空白视图。

乔：嗯，我们需要添加两个表格视图，还有用来在这些视图之间切换的标签栏导航控制器，以及细节视图。

弗兰克：那是不是我们需要一大堆新的nib文件来存放这些视图和控制器呢？

吉姆：唉。这个基本模板对我们基本没用！

乔：没那么糟糕啦。我觉得可以把它看成是一张能够随意挥洒的白纸。让我想想，我们可以从标签栏和标签栏控制器开始……

弗兰克：对啊，这个控件可以在“逃犯”和“捕获”这两个表格视图之间切换。这些视图还各自需要一个导航控制器来进出细节视图。

乔：那我们需要为标签栏和这两个视图准备单独的nib吗？看起来，好像我们能把标签栏和这两个视图用到的控件全放进一个nib里面去，因为它们看起来基本是一样的。

吉姆：是啊，但我们仍然需要为每个视图单独准备视图控制器、头文件和.m文件。

乔：没错，它们这些视图里面都得有个表格。我们还需要一个细节视图，也得有自己的nib和视图控制器，以及.h和.m文件，我说的没错吧？

弗兰克：这听起来没什么问题。我们可以用Interface Builder来创建标签栏和导航控制器。

乔：那剩下的东西怎么办？我们要在Xcode里加新文件吗？

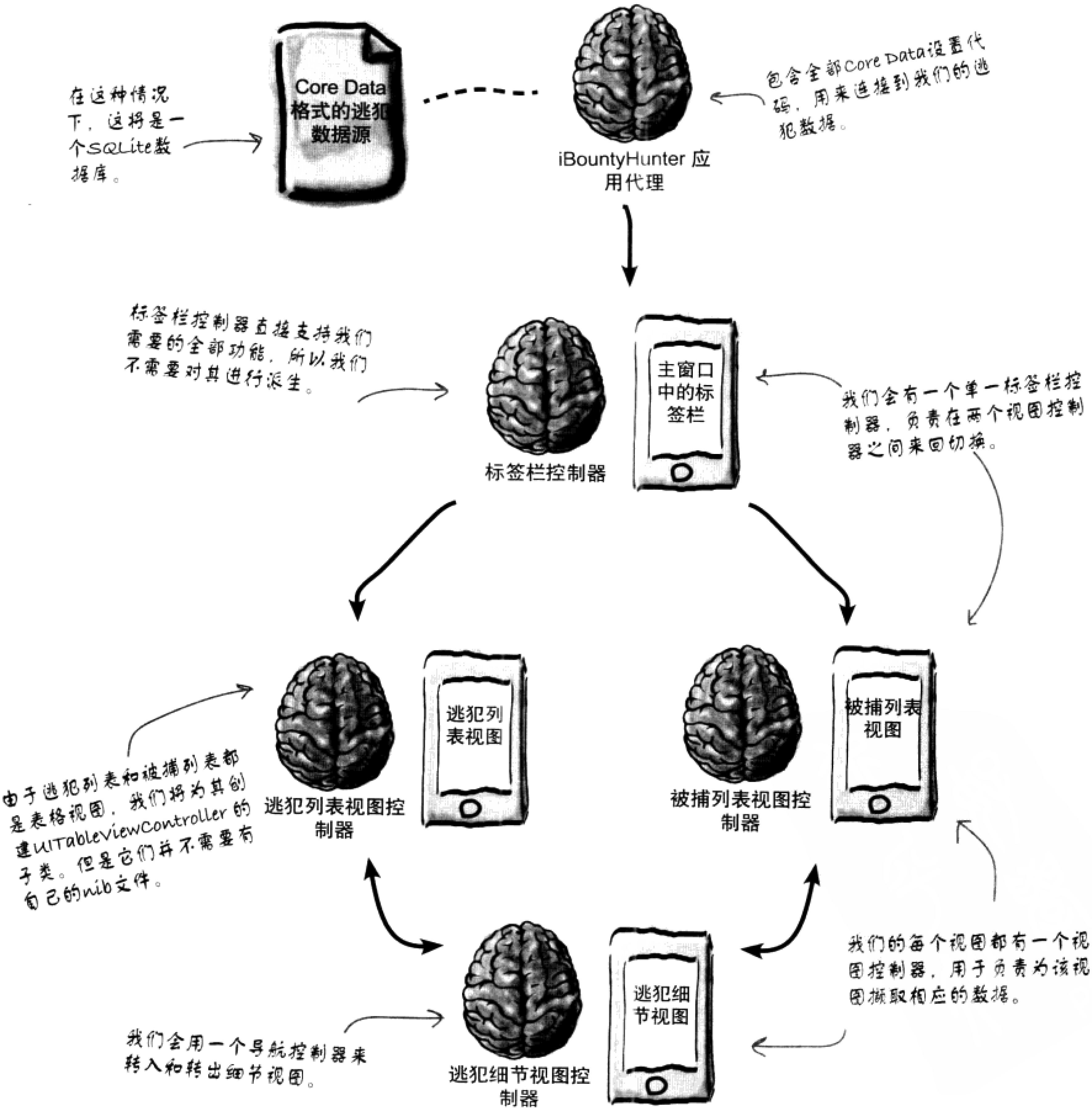
弗兰克：这样也行——就和以前一样，我们只要指明nib文件也要同时创建出来，然后应该就没问题了。

吉姆：我觉得这些都可以——得处理好多东西啊。

乔：好吧，我们现在要把三种不同的东西组合起来，所以肯定会变得更复杂的！可能画画图会有帮助，看看怎么把这些东西全弄在一起？

【拼吾爱-最新编程资源的分享下载站】

画出iBountyHunter的工作过程……



【拼吾爱-最新编程资源的分享下载站】



乔：这帮助很大。这样来说我们只需要两个nib，一个当中存放通过标签栏切换的逃犯和被捕这两个视图用到的控件，另一个负责管细节视图。

弗兰克：我明白了。我们需要将表格视图的组件放在一个地方，然后我们可以为每个视图创建一个新nib，让标签控制器加载它们……

吉姆：……也可以把它们全都放在一个nib里。太简单了！

弗兰克：正是。由于我们不打算在其他地方重用这些表格视图，而它们也不是太复杂，所以我们可以只用一个nib，让事情变得简单一点。

吉姆：而我们需要为两个表格视图准备视图控制器，还有细节视图。它们将会负责根据用户所处的视图来获取正确的数据。

弗兰克：还要加上一个导航控制器，好让表格视图能转入、转出细节视图。

乔：我认为我们可以开始构建应用了！

iBountyHunter待办事项列表

1. 为逃犯与被捕视图创建视图控制器（.h和.m文件）。
2. 创建标签栏视图，并为之添加标签栏控制器，以及一个对应用代理的引用。
3. 为逃犯与被捕视图添加导航控制器。
4. 为逃犯与被捕视图构建表格视图。
5. 创建一个细节视图nib，以及视图控制器的.h和.m文件。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问： 我们为什么要使用标签栏控制器和表格视图？

答： 我们的逃犯数据是分层次的，很适合用表格视图来表示。问题在于，我们有两个表格视图：逃犯列表和被捕列表。所以为了支持两个顶级列表，我们选择了标签栏。

问： 你不能用一些类似切换开关的东西吗？比如UISegmentControl。

答： 是的，我们可以这样做。这其实只是一个UI设计上的选择问题。这两个列表其实是不同的列表，而不是对相同数据的不同排列组织方式。不过，这些都是主观认定的。

问： 哎呀，我对给标签控制器和两个表格视图只准备一个nib的事，始终有点迷惑。

答：是啊，这个应用里面有很多事情要做，而我们本来可以用另外一种方法来完成这些任务。我们本来可以再多创建两个nib，每个里面都配上一个导航控制器和一个表格视图，那样就可以让标签栏加载第一个nib作为逃犯列表，将第二个作为被捕列表。然而我们并没有这样做，相反，我们将列表用到的这些控件放进标签栏使用的同一个nib中。记住，nib只不过是容纳了UI控件，并不控制它们的行为。

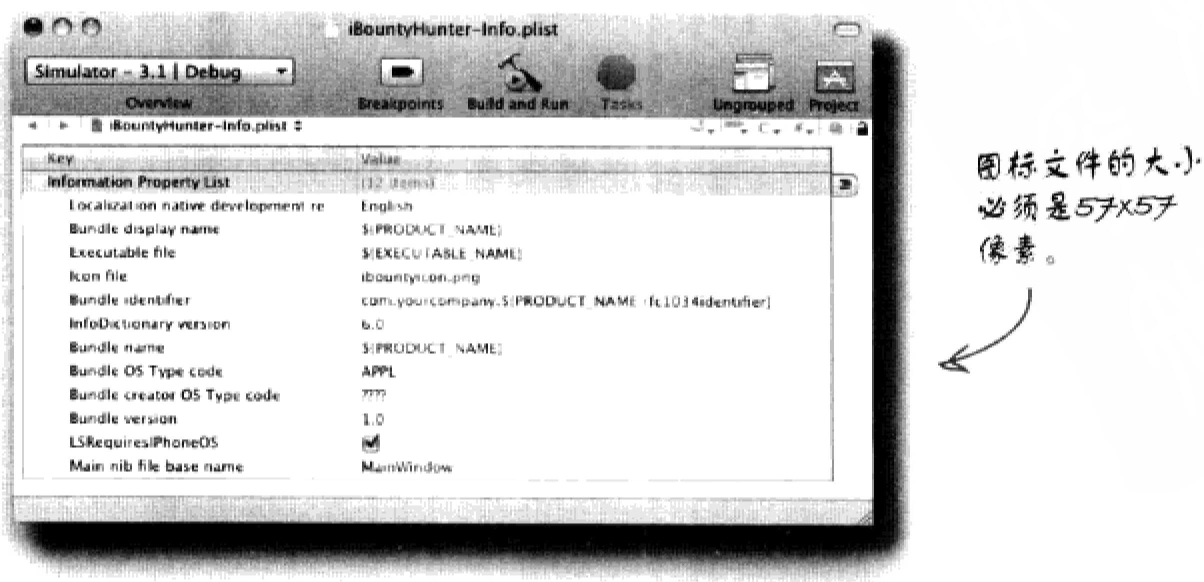
问： 不过，说实话——这个办法真的要比使用标签栏模板、然后再按照我们的需要进行调整来得好吗？

答： 那种办法的确也是一种选择。然而，如果我们审视标签栏模板就会发现，它自带了两个分支，其中一个被单独存放到一个nib中，这是为了向你展示这样做的可行性，而另一个则在同一个nib中（这也是为了展示一种可行性）。所以我们必须修改其中的一个，或者维持这种分裂的方式，后面这种选择很快就会变得越来越恶心。我们还不得不去修改非常大量的默认配置，其中的半数都在另一个nib中，另一半则是内嵌的。总之，我们采取的这种方法比前面的复杂程度还要小一点，而且是建立在你目前学过的方法之上的。

按图索骥！

为你的应用加一个图标。

你马上就要开始激情奉献大量代码了。在沉浸其中之前，请先访问一下网址http://www.headfirstlabs.com/iphonedev，下载iBountyHunter的图标（ibountyicon.png），并将它拖放到你的新项目的**/Resources**文件夹中。然后在Xcode中打开iBountyHunter-info.plist，将文件的名字输入到图标这一栏。



【拼吾爱-最新编程资源的分享下载站】



练习

在Xcode中为逃犯视图和被捕视图创建两个新类，然后在Interface Builder中添加你的标签栏控制器。

1. **创建两个新类的.m和.h文件。**
   这些文件是逃犯列表和被捕列表的视图控制器。FugitiveListViewController.h与.m以及CapturedListViewController.h与.m均需是UITableViewController的子类，因此在生成文件时请选择“UIViewController subclass”，同时要选中 UITableViewController子类选项。

2. **添加标签栏控制器。**
   在Interface Builder中，首先打开MainWindow.xib，然后将标签栏控制器拖放在视图中。

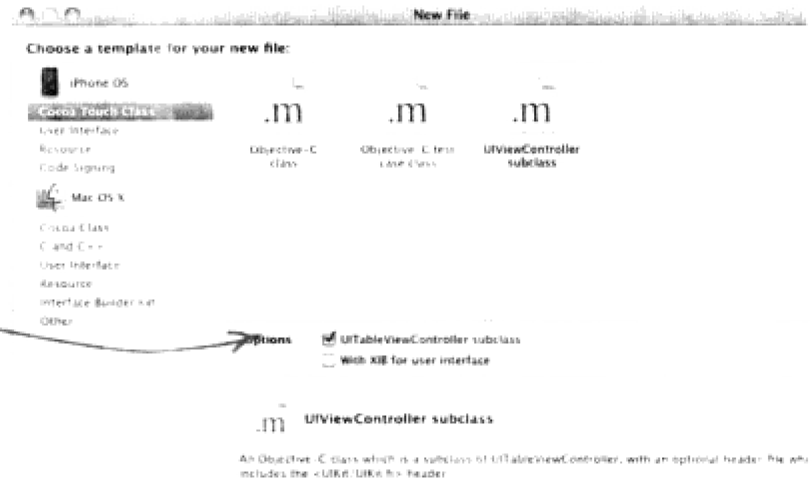
【拼吾爱-最新编程资源的分享下载站】



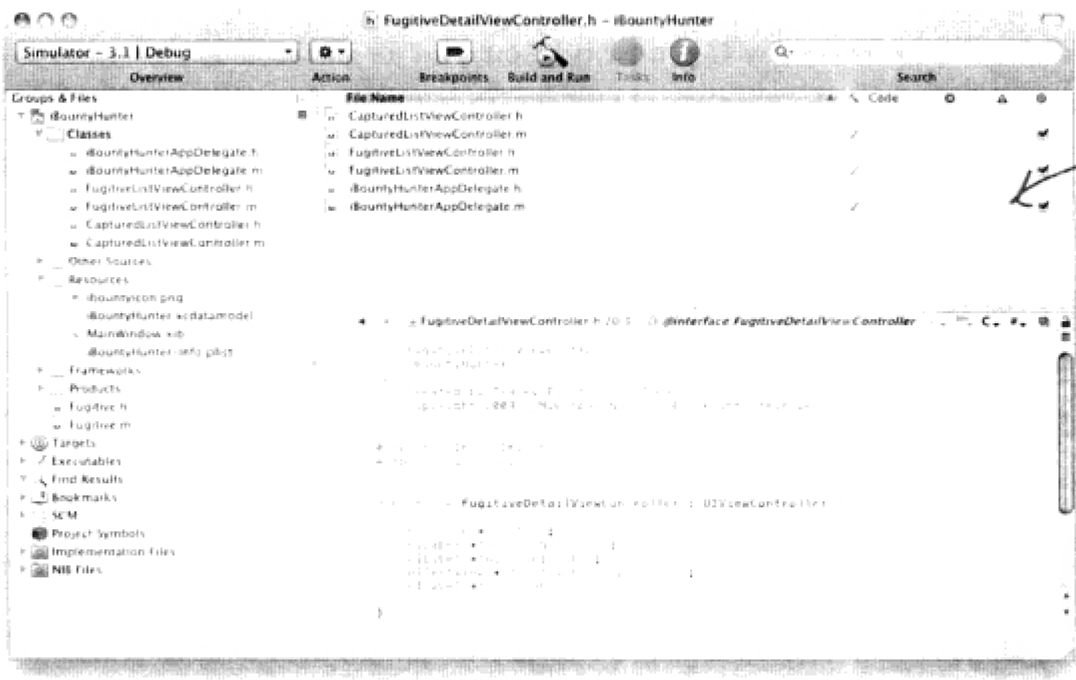
练习
答案

在Xcode中为逃犯视图和被捕视图创建两个新类，然后在Interface Builder中添加你的标签栏控制器。

1 创建两个新类的.m和.h文件。



创建这些文件时，要确保它们是UITableViewController的子类，而且没有选中那个"With XIB for user interface"框。



当你完成之后，文件列表看起来应该就像这样。



留神！

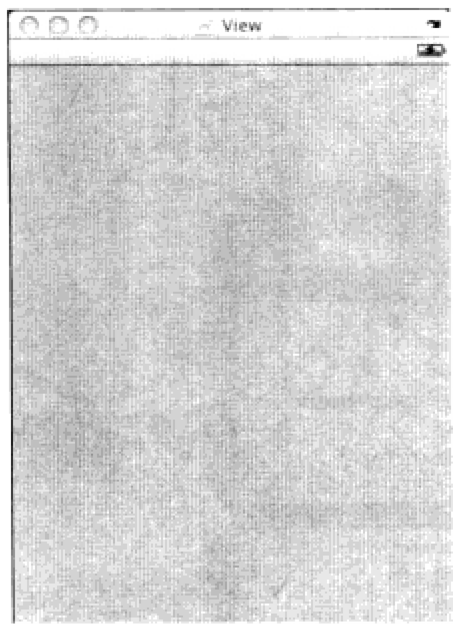
在Xcode 3.1里面你不会见到UITableViewController点选框！

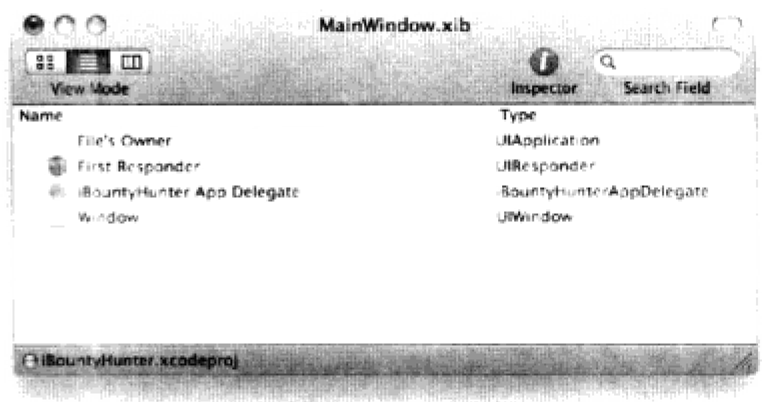
如果你用的不是XCode 3.2（Snow Leopard），就需要进入CapturedListViewController.h和FugitiveListViewController.h文件当中，将它们从UIViewController的子类改为UITableViewController的子类。

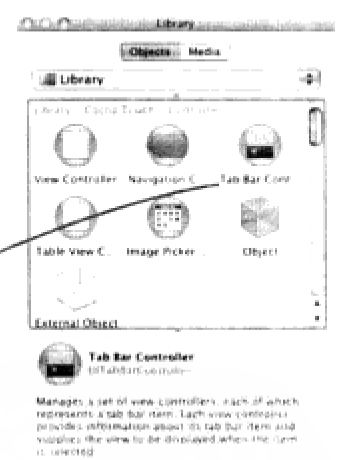
❷ 添加标签栏控制器。

窗口模板中并未带有太多内容。我们打算用Interface Builder按照我们需要的方式来组建自己的视图和视图控制器。

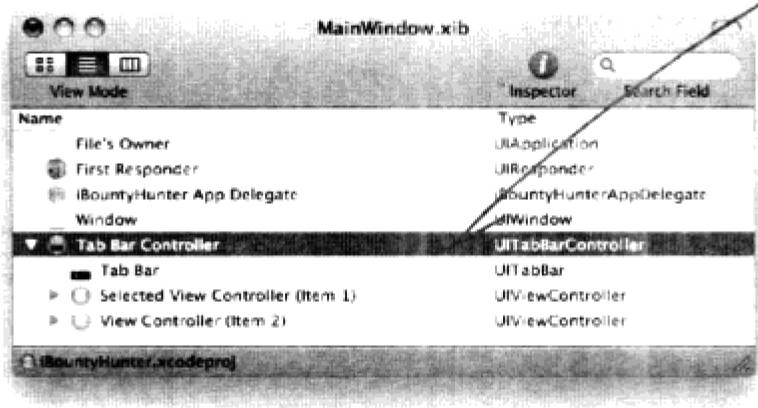
模板自己只带有一个空白的UIWindow窗口。这就是我们的应用代理在启动时将会显示的窗口。

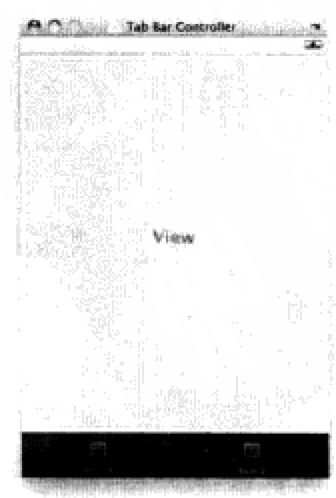






将标签栏控制器从控件库中拖拽到你的主窗口列表中。这样就会为你创建标签控制器视图：





标签栏控制器自带一个标签栏和两个内建视图控制器，但是我们马上就会对其进行改动…

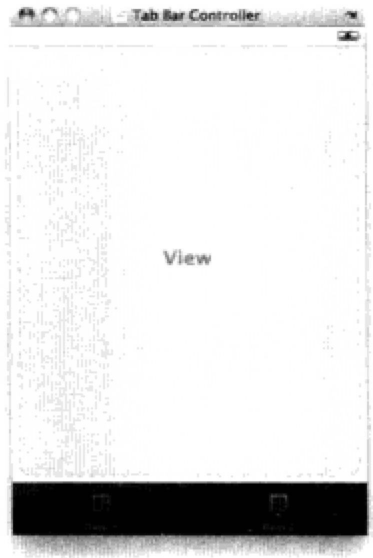
【拼吾爱-最新编程资源的分享下载站】

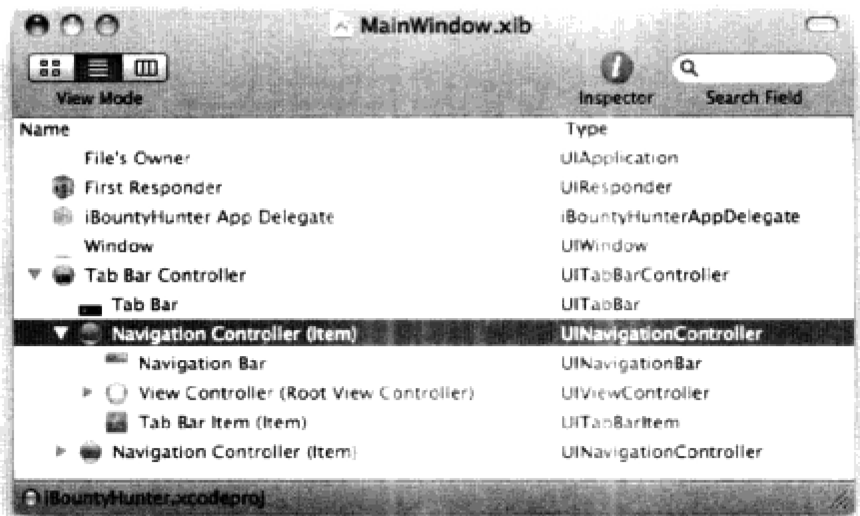
构建逃犯列表视图

首先我们将集中关注逃犯列表，但是相同的步骤也可以用于被捕列表。

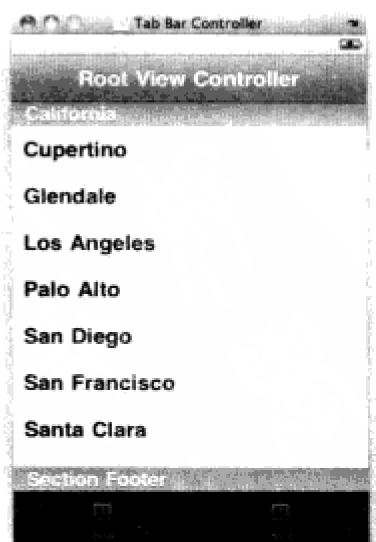
❶ **删除那两个视图控制器，并代之以导航控制器。**
由于我们需要导航控制器的全部功能，所以请删除那些视图控制器，并从库中拖放两个新的导航控制器来代替它们。要确保它们被放在标签栏控制器的下面。

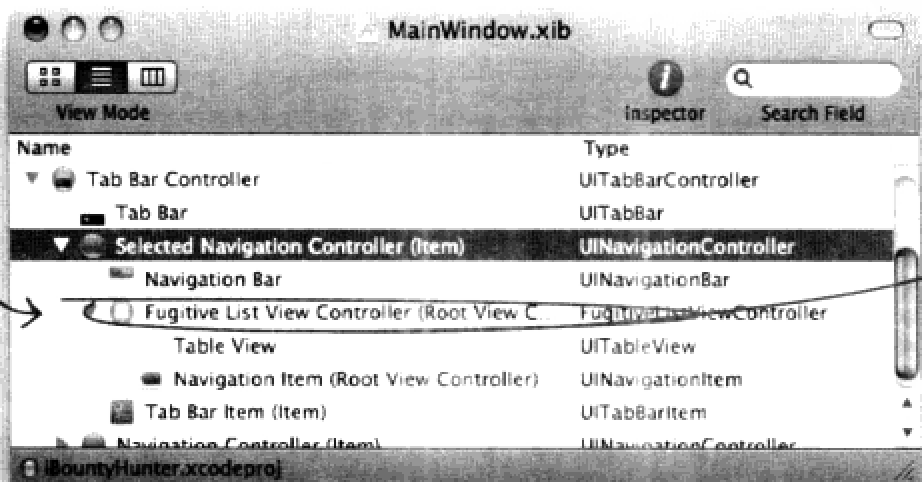
在视图中没有任何变化——只有主窗口的列表反映出你更改过的东西





❷ **将视图控制器更改为逃犯列表视图的控制器。**
选中第一个导航控制器下面的视图控制器，用组合键⌘4将其Class属性改为FugitiveListViewController。



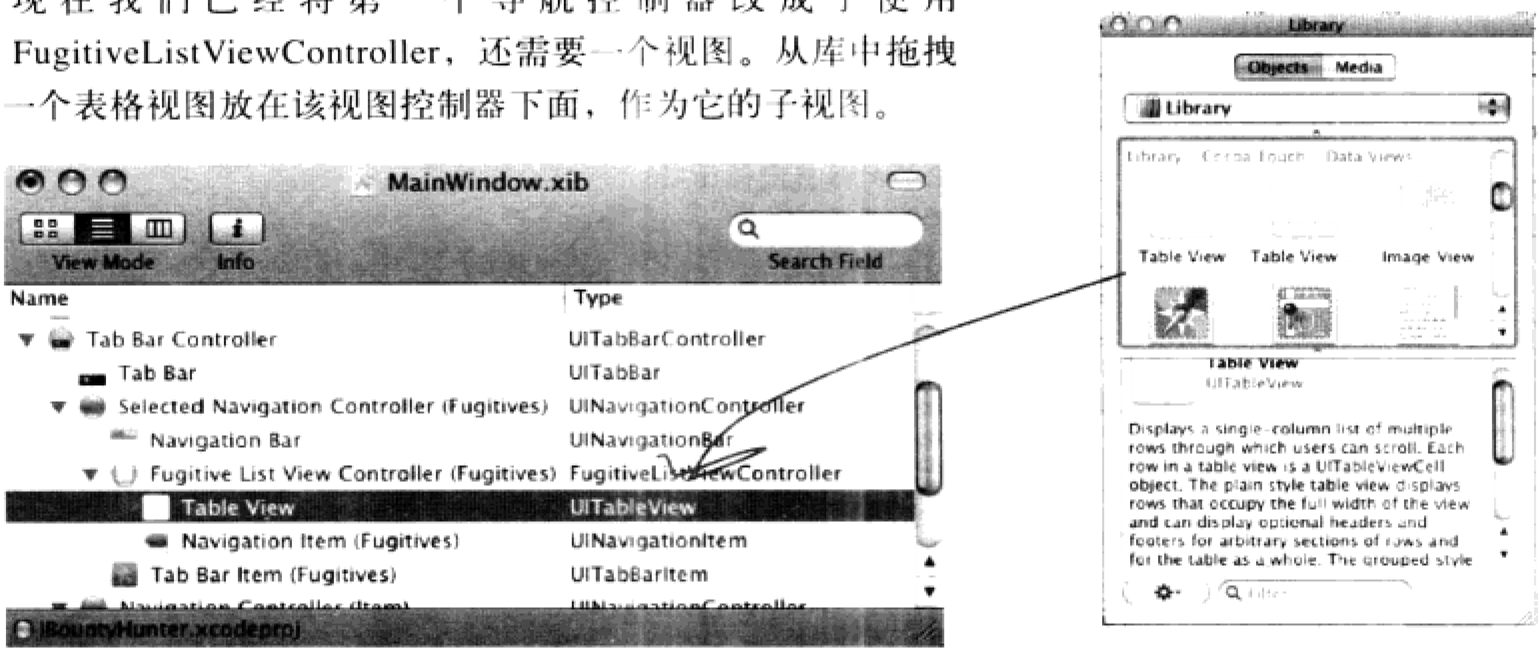


导航控制器自带了一个默认UIViewController。我们不想要这个默认的，而要让导航控制器使用我们的逃犯列表视图控制器。

【拼吾爱-最新编程资源的分享下载站】

❸ 添加表格视图。

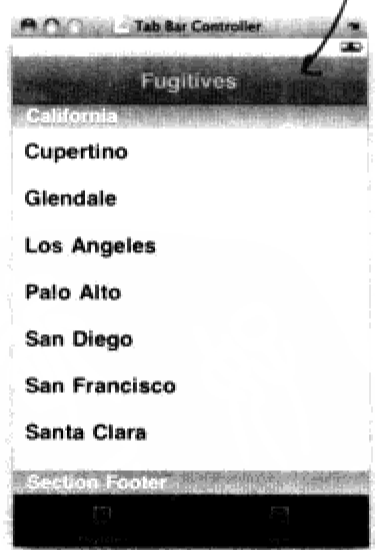
现在我们已经将第一个导航控制器改成了使用FugitiveListViewController，还需要一个视图。从库中拖拽一个表格视图放在该视图控制器下面，作为它的子视图。

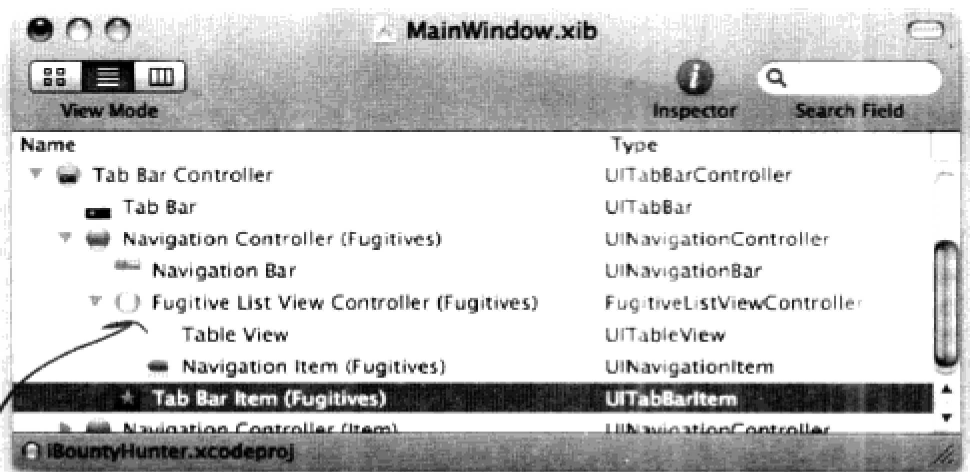


❹ 设置标签栏和导航栏中的名称。

要更改逃犯列表视图控制器的标题，可以双击导航栏上的标题，然后输入“Fugitives”(逃犯)。对于标签，可以单击第一个条目，按下⌘1，将属性 Bar Item Title改为“Fugitives”。

单击这里可以修改视图控制器的标题。





修改后的标签条目标题。

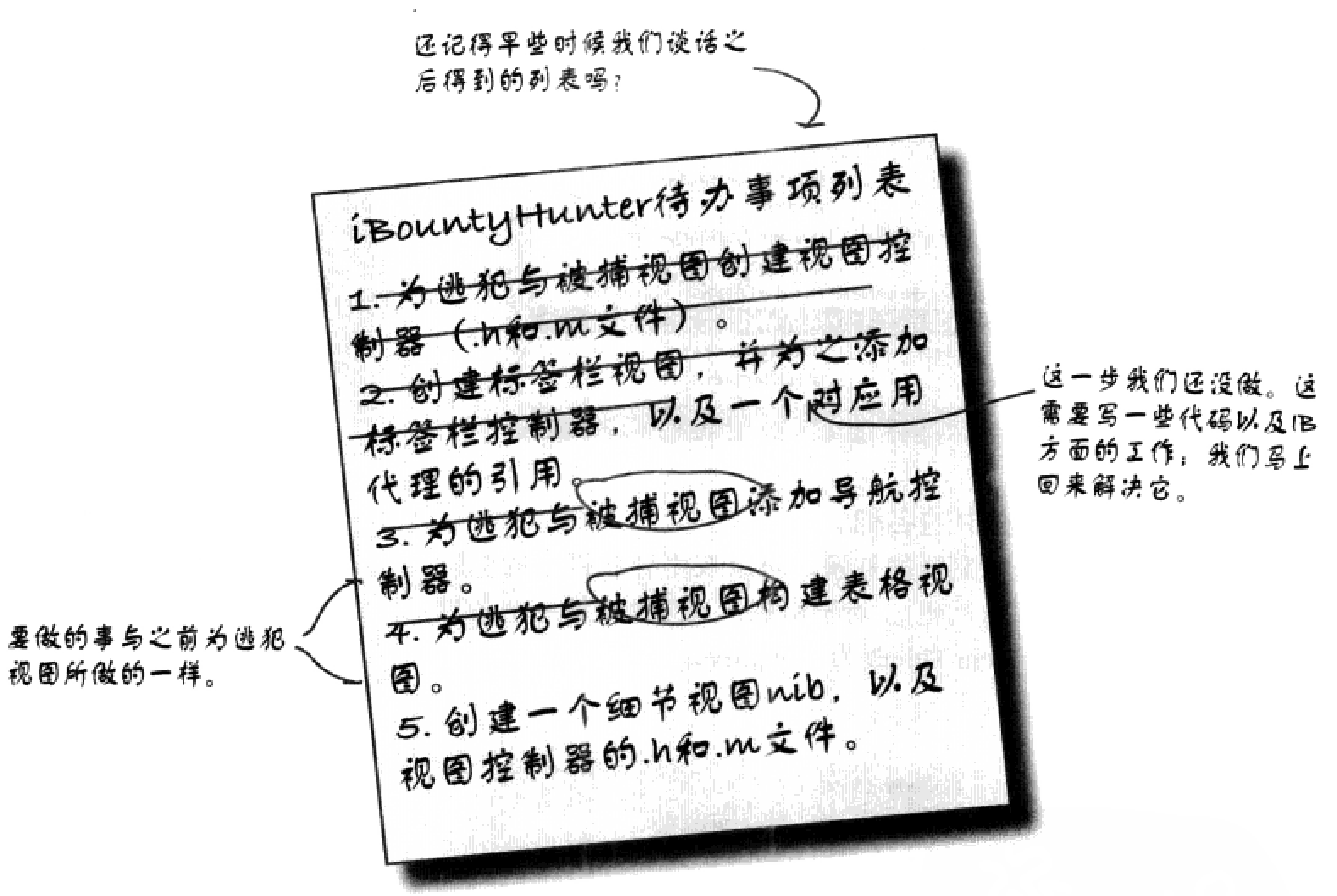
修改后导航控制器标题也反映在副标题上。

接下来做什么？

【拼吾爱-最新编程资源的分享下载站】

接下来：被捕视图

你已经完成了为两个表格视图创建类的工作，并加入了一个标签控制器，以便在两个视图之间切换。



【拼吾爱-最新编程资源的分享下载站】

人人都是开发者



你的任务是作为一个开发者，在Xcode和Interface Builder中完成剩下的工作，让逃犯视图和被捕视图能够在标签栏控制器中使用。利用吉姆、弗兰克和乔列出的待办事项列表，找出剩下该做的工作。

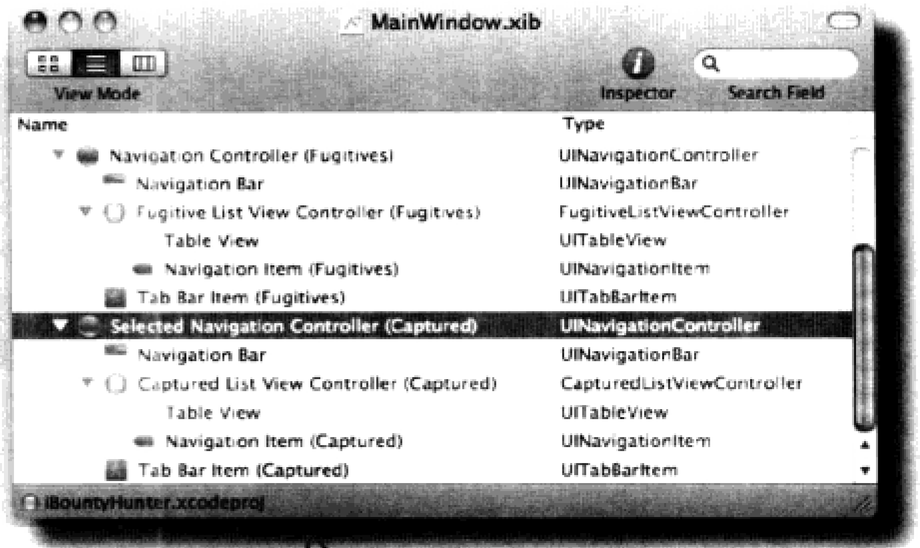
创建被捕视图就靠你了，而且还要将视图与标签栏控制器联系起来……

【拼吾爱-最新编程资源的分享下载站】

人人都是开发者



你的任务是作为一个开发者，在Xcode和Interface Builder中完成剩下的工作，让逃犯视图和被捕视图能够在标签栏控制器中使用。利用吉姆、弗兰克和乔列出的待办事项列表，找出剩下的该做的工作。



最后你得到的列表看起来应该就像这样。

创建你的被捕视图。

按照之前创建逃犯视图时相同的步骤。

然后与标签栏控制器连接起来。

要做到这一点，我们需要回头来看看应用代理。目前还不存在一个输出口，让我们能够将标签栏控制器与其他东西联系起来，所以它无法正常工作。现在你应当已经对如何完成这项工作相当熟悉了。下面就是标签栏控制器需要的输出口：

```
  UIWindow *window;
UITabBarController *tabcontroller;
```

```
@property
(nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UITabBarController
*tabcontroller;
```

iBountyHunterAppDelegate.h

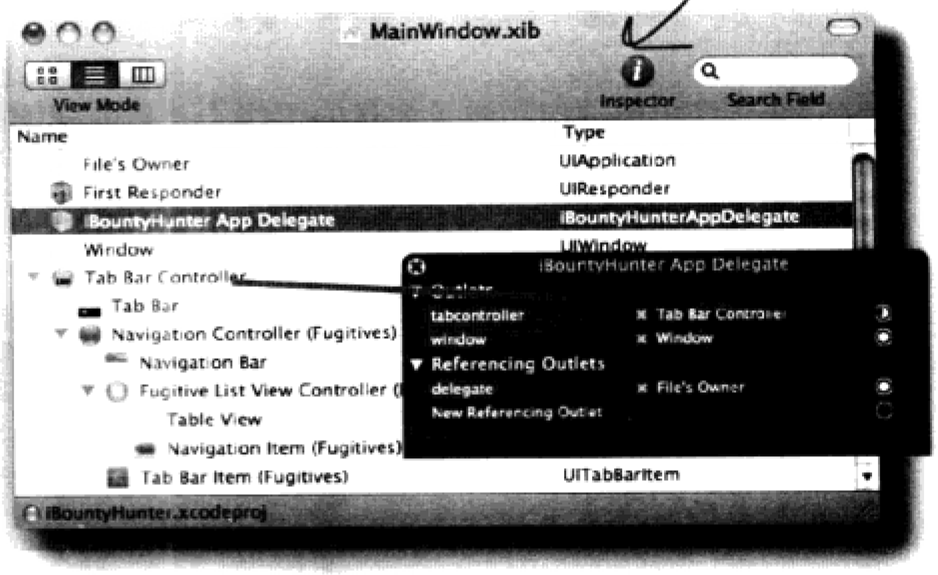
差不多了……

```
@synthesize window;
@synthesize tabcontroller;
```

iBountyHunterAppDelegate.m

```
[persistentStoreCoordinator
release];
[tabcontroller release];
```

我们需要在这里将应用代理与标签栏控制器连接起来。



有问必答

问：我们塞了很多东西在主窗口的nib里面。它看起来还是让我觉得有点奇怪。

答： iBountyHunter的nib中包含了5个控制器（标签栏、两个导航控制器以及我们的FugitiveListViewController和CapturedListViewController）以及与它们相关的组件。如果你仍然弄不清楚，请在Interface Builder中打开MainWindow.xib文件，以树形模式来观察其结构，应该有所帮助。将里面的层次展开，就可以看到我们的应用的结构。我们拥有一个单一nib，里面有一个标签栏控制器，其内部嵌套有两个导航控制器，分别是FugitiveListViewController与CapturedListViewController的实例。

问：我能在标签栏上添加图标吗？

答： 绝对没问题。最简单的方法就是用Interface Builder挑选一个标准图标。要使用这种方法的话，请点击要进行替换的标签上面的问号图标，然后在查看器中修改Identifier属性。如果你想要使用自定义图像，可以将Identifier设为自定义，然后在Image字段中选择你的图像（得把图像添加到项目中，就像以前我们对应用程序图标做的那样）。不过，对于标签栏图标有一些不同寻常之处：它们的大小应该是30×30，而且实际上是利用图标的透明度alpha值来生成图像的。你并不能指定颜色或者其他类似的东西。

问：标签栏中可以有多少个视图？

答： 多少都行。如果加入的视图太多，在屏幕底部的标签栏中放不下，UITabBarController就会自动添加一个标有“更多”的标签，点击它就会在一个表格视图里显示剩下的内容。默认情况下，UITabBarController还包括了一个编辑按钮，让用户可以编辑哪些标签会显示在底栏上。

问：有没有什么办法可以知道用户何时切换标签？

答： 有办法，你可以编写一个遵循UITabBarDelegate协议的类，并将其设为标签栏的代理。这样你就会在用户对底栏进行定制或者切换标签时收到通知。

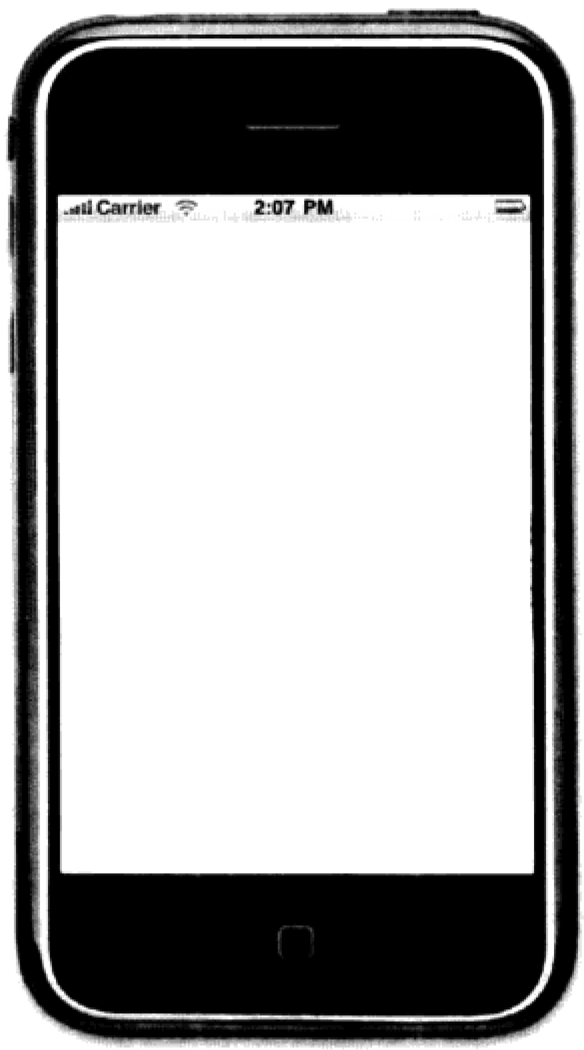
问：为什么我们要在应用代理中添加一个对标签栏控制器的引用？

答： 我们已经在nib中添加了标签栏控制器，但是要想让一切正常地显示出来，还有一些额外的调整工作要做。试验一下后面的内容，看看会出现什么情况……

【拼吾爱-最新编程资源的分享下载站】

运行测试

你已经为你的应用做了大量的工作——新的视图控制器、新的导航控制器、表格视图——全都是从头做起。构建并运行应用，确认一下一切正常。



啊呀！什么也没有！为什么没出现标签栏控制器（还有其他的东西）？

【拼吾爱-最新编程资源的分享下载站】

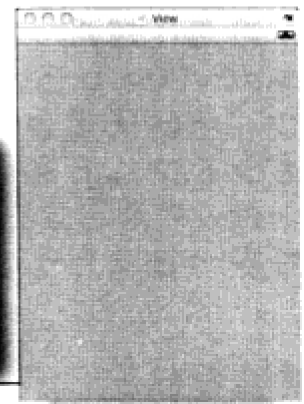
拿起笔来

想办法找出问题的原因。回顾一下我们之前在Interface Builder中的工作，看你能不能发现哪里做错了……

❶ **首先打开MainWindow.xib。**

我们一开始用的模板是一个空白窗口，但列表不是空的。

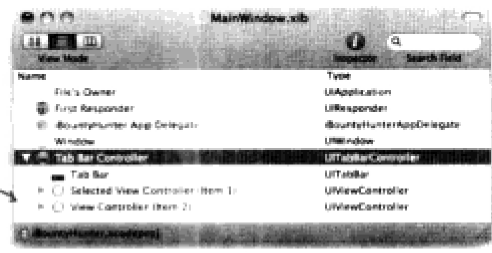




❷ **添加标签栏控制器。**

因为它要负责使用那两个视图，所以我们从这里开始。

标签栏控制器自带了一个标签栏和两个内建的视图控制器，不过它们不会存在太久。





【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

找出问题的原因。回顾一下我们之前在Interface Builder中的工作，看你能不能发现哪里做错了……

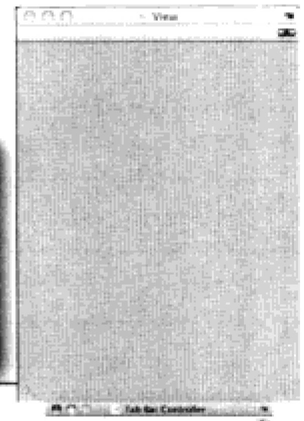
问题在于，标签栏在nib中是一个顶级元素。应用代理将UIWindow作为自己的窗口，所以代理会显示这个窗口。但是UIWindow并未包含任何内容——它没有子视图。

我们需要将标签栏控制器嵌入到UIWindow中。

1 首先打开MainWindow.xib。

我们一开始用的模板是一个空白窗口，但列表不是空的。





2 添加标签栏控制器。

因为它要负责使用那两个视图，所以我们从这里开始。





标签栏控制器自带了一个标签栏和两个内建的视图控制器，不过它们不会存在太久。

一个视图的内容实际上就是子视图

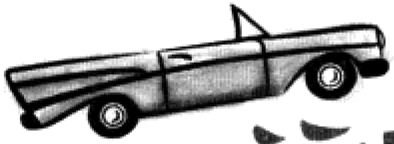
我们使用过的所有UI组件都是UIView的子类。我们将它们拖放到视图中，就是将其变成了这些更大的容器视图的子视图。我们需要对标签栏做相同的处置；然而问题在于，我们无法在Interface Builder中得到UIWindow的视图。我们得在代码中完成这项工作。

在Interface Builder里能做到的事，在代码里也都能做到。

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // Override point for customization after app launch
    [window addSubview:tabcontroller.view];
    [window makeKeyAndVisible];
}
```

我们需要将标签栏变成UIWindow的子视图。你可以在ApplicationDidFinishLaunching方法中这样做。

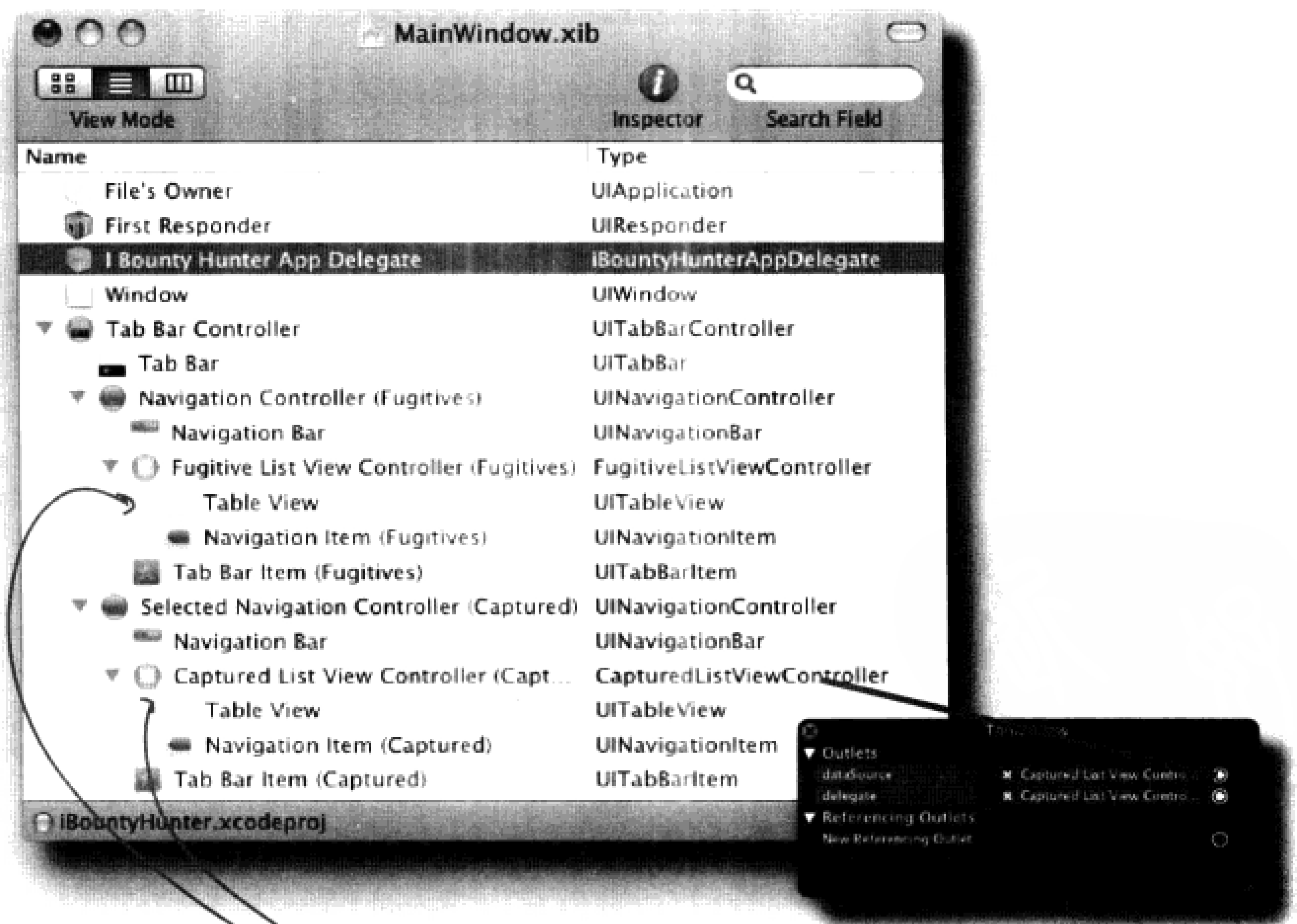
iBountyHunterAppDelegate.m



运行测试

快要成功了。只需要在Interface Builder里撮合几个连接关系，我们就可以完工了。

表格视图也需要与两个视图控制器建立联系，还有就是应用代理中的输出口要与逃犯控制器和被捕控制器连接起来。

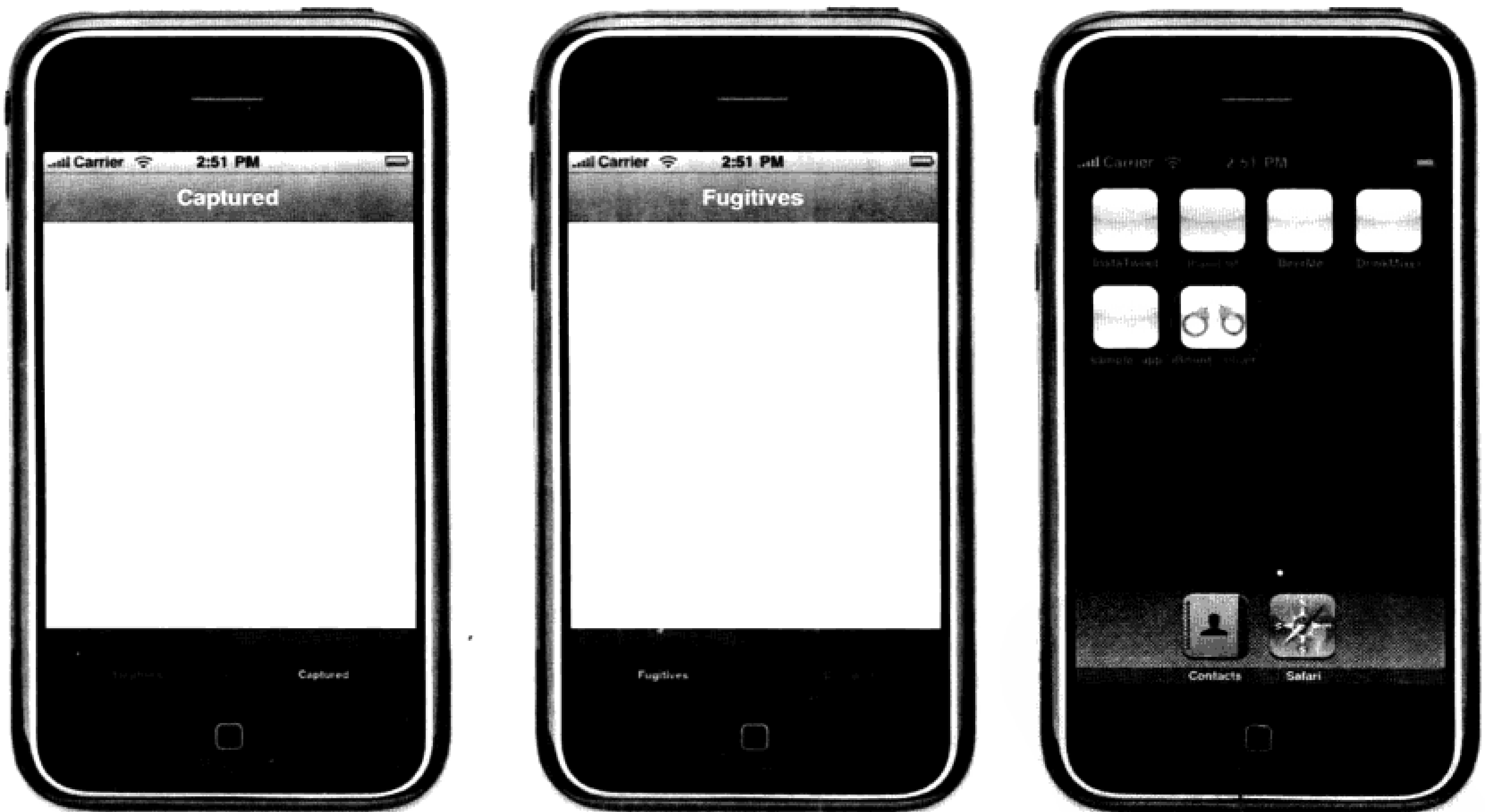


对于表格视图，代理和数据源也都要与它们的父视图控制器联系起来。

【拼吾爱-最新编程资源的分享下载站】

运行测试

现在应该看到一切工作正常了。构建并运行应用，你会看到两个表格视图都正常地显示出了表格。



还记得我们之前安装的那个图标吗？来吧，按下主按钮看看它是否显示正常。

【拼吾爱-最新编程资源的分享下载站】

在与鲍勃开了个短会之后……

目前看起来不错。这是我的逃犯列表。这张列表目前还是很老式的——只是一份法庭发出的打字的清单。



下一步就是管理鲍勃的数据。

现在应用已经初步建立并运行起来了，你需要填补剩下的空白。目前这个列表相当简单，所以我们可以将数据任意转换成所需格式，然后将其导入。



脑力练习

我们该怎样表示这些数据?

【拼吾爱-最新编程资源的分享下载站】



弗兰克：我想了想——不是很确定这次还用plist好不好。

吉姆：为什么不好？我们在DrinkMixer里就是用的这个，用起来挺不错的啊。

弗兰克：嗯，这个列表可能会很大——记得吗，逃犯列表会始终处于变化中：一些是鲍勃试图抓获的，还有那些已经抓获的。

乔：那怎么了？

弗兰克：那么……一个巨大的列表也就意味着需要大量内存。

乔：哦，就是啊——plist每次都会把所有东西都加载进来。

弗兰克：正是。

吉姆：用那个Core Data怎么样？这个应该是用来处理大量数据的，对吧？

弗兰克：这是新的3.x系统里的数据框架。八成应该可以。

吉姆：为什么用这个，而不是简单地用一个数据库？iPhone不是支持SQLite吗？

弗兰克：确实支持，不过我对SQL不熟，而Core Data可以支持所有种类的数据，也包括SQL，但是你不必与其直接对话。

乔：你的意思是我们不用SQLite？

弗兰克：我们用，但是会用Core Data来访问它。

乔：怎么个用法呢？

弗兰克：Core Data替我们把脏活累活都干了，我们只需要告诉它想要加载和保存什么数据就行了……

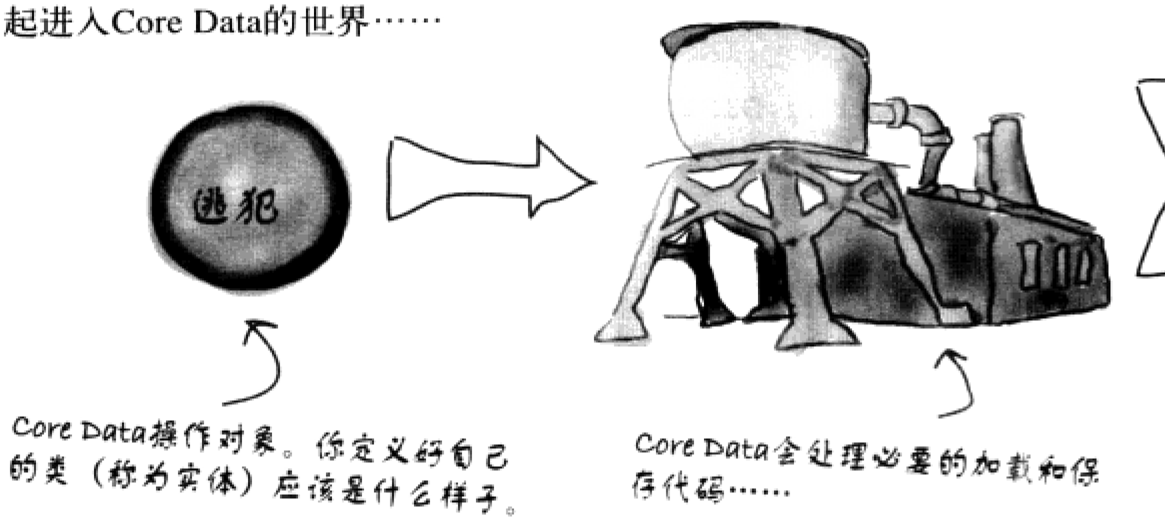
脑力练习

我们在DrinkMixer中用到了plist和字典来储存数据，这种方法还有其他哪些限制？

【拼吾爱-最新编程资源的分享下载站】

Core Data让你专注在应用上

加载与保存数据，特别是大量的数据，对于大多数应用程序来说都是一个重要部分。我们已经花费了大量的时间来熟悉plist以及如何将对象从数组中移入移出。但是如果你想要以多种不同的方式来排序数据，或者只想看到赏格超过一百万美元的逃犯，或者要处理10万个逃犯的数据，该怎么办呢？为处理这种持久化数据而编写的代码已变得很成熟、快捷。让我们一起进入Core Data的世界……





但是等一下，还有一些东西！

Core Data令加载与保存数据变得十分方便快捷，但它并不止于此。它是一个成熟的框架，在iPhone 3.0版本中，苹果公司将它从Mac OS X带到这个平台，随之而来的是：

- **保存和加载对象的能力**
  Core Data可以根据实体的描述自动加载和保存对象。它甚至可以处理对象之间的关系、在不同版本的数据之间迁移、识别必须和可选字段以及进行字段校验。

- **多种不同的数据存储方式**
  Core Data向你的应用程序隐藏了数据实际存储的方式。你可以从一个SQLite数据库或者一个自定义二进制文件中读写数据，只要简单地告诉Core Data，希望它如何保存你的内容，就可以自动完成。

- **具备撤消和重做功能的内存管理**
  Core Data能够极度高效地对内存中的对象进行管理，并跟踪对象的变化。你可以用它来进行撤销和重做，将巨量的信息从数据库中分页切换进出……此外还有很多功能。

但在我们做到任何一点之前，需要将对象的信息告知Core Data……

【拼吾爱-最新编程资源的分享下载站】

Core Data需要知道如何进行加载

我们需要让Core Data加载与保存逃犯信息，而且要填充我们的细节视图。如果你回想一下DrinkMixer，应该记得我们使用字典来保存饮料信息，用键值来对它们进行访问，如下所示：

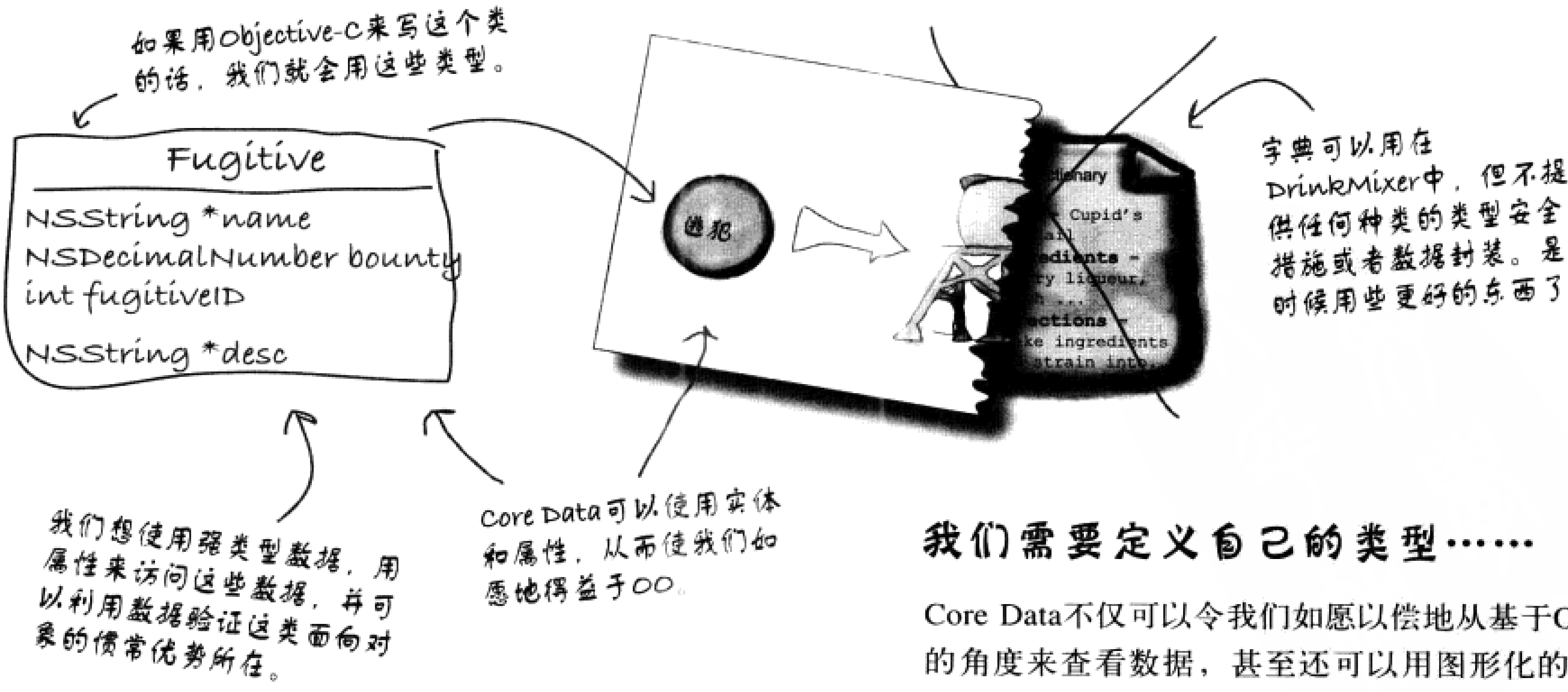
```
nameTextField.text = [drink objectForKey:NAME_KEY];
ingredientsTextView.text = [drink objectForKey:INGREDIENTS_KEY];
directionsTextView.text = [drink objectForKey:DIRECTIONS_KEY];
```



DrinkMixer用字典来储存饮料信息。可以用，但是相当原始。

用Core Data来填充这些数据。

用字典和plist的问题在于，我们必须用基本类型来储存所有数据，并且通过字典的键值来获取数据。这样很容易出现错误，比如将错误的类型放入了字典中，或者使用了错误键值，从而引起许多后续问题。我们真正需要使用的是正常的Objective-C类和对象，好让我们可以为成员变量声明属性，使用真正的数据类型，等等。这些正是Core Data可以让我们做到的。



我们需要定义自己的类型……

Core Data不仅可以令我们如愿以偿地从基于OO的角度来查看数据，甚至还可以用图形化的方式来定义数据。不过有一个内在的障碍，Core Data支持的是一组特定的数据类型，所以我们需要用它提供的数据来定义实体……

【拼吾爱-最新编程资源的分享下载站】

对号入座

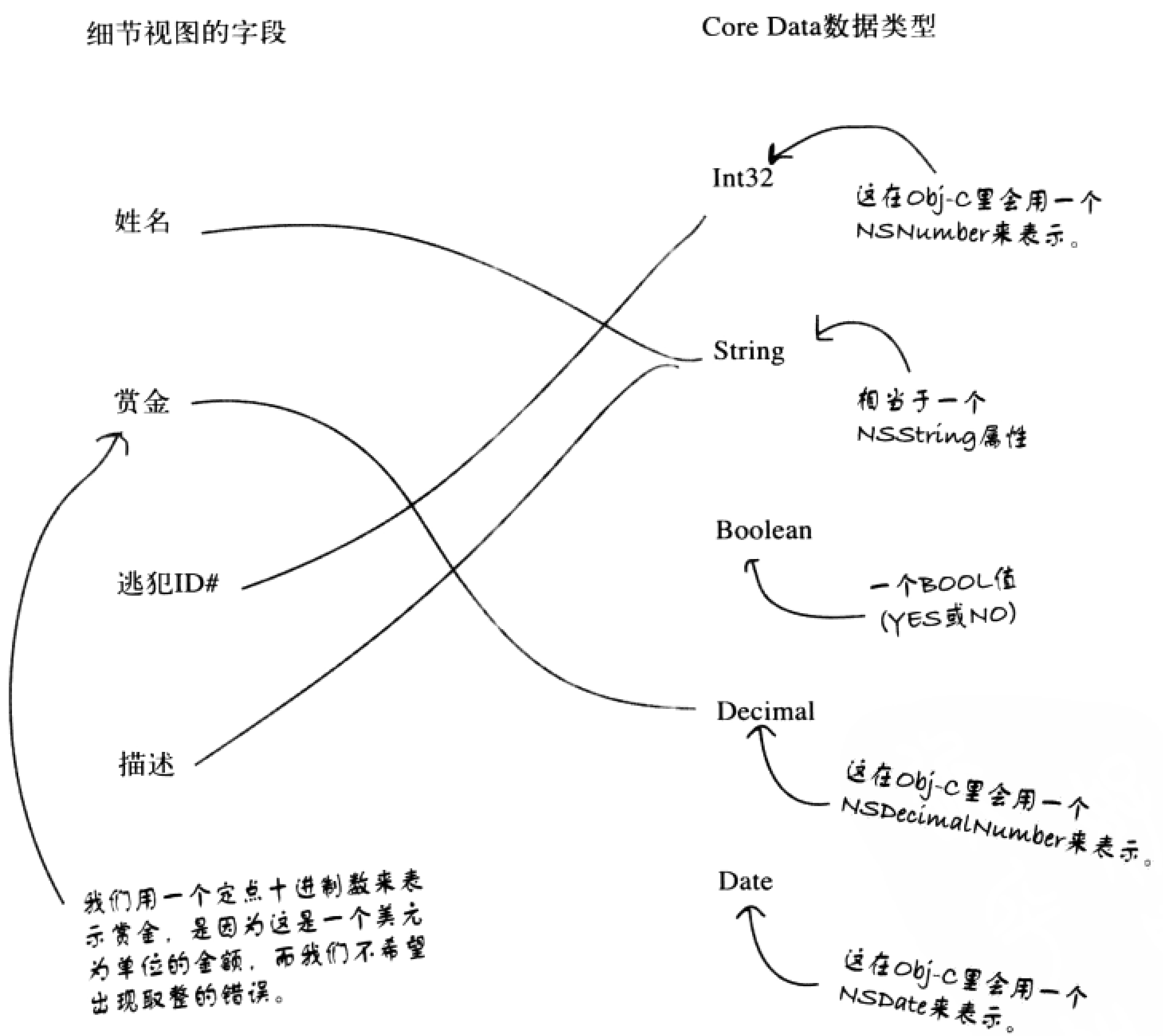
将我们需要为数据视图实现的每个字段，与它的Core Data数据类型对应起来。

细节视图的字段	Core Data数据类型
姓名	Int32 ← 一个32位整数
赏金	String ← 相当于一个NSString属性
逃犯ID#	Boolean ← 一个BOOL值（YES或NO）
描述	Decimal ← 一个小数点后位数固定的十进制数
	Date ← 日期与时间信息

【拼吾爱-最新编程资源的分享下载站】

对号入座 答案

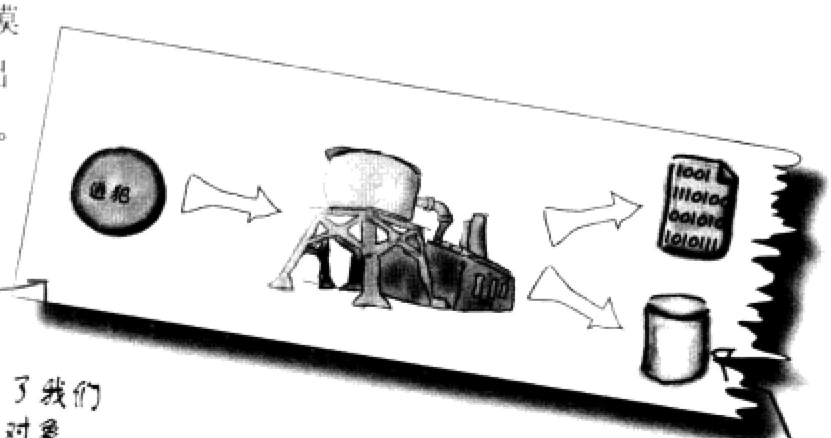
将我们需要为数据视图实现的每个字段，与它的Core Data数据类型对应起来。



【拼吾爱-最新编程资源的分享下载站】

Core Data用一种受控对象模型来描述实体

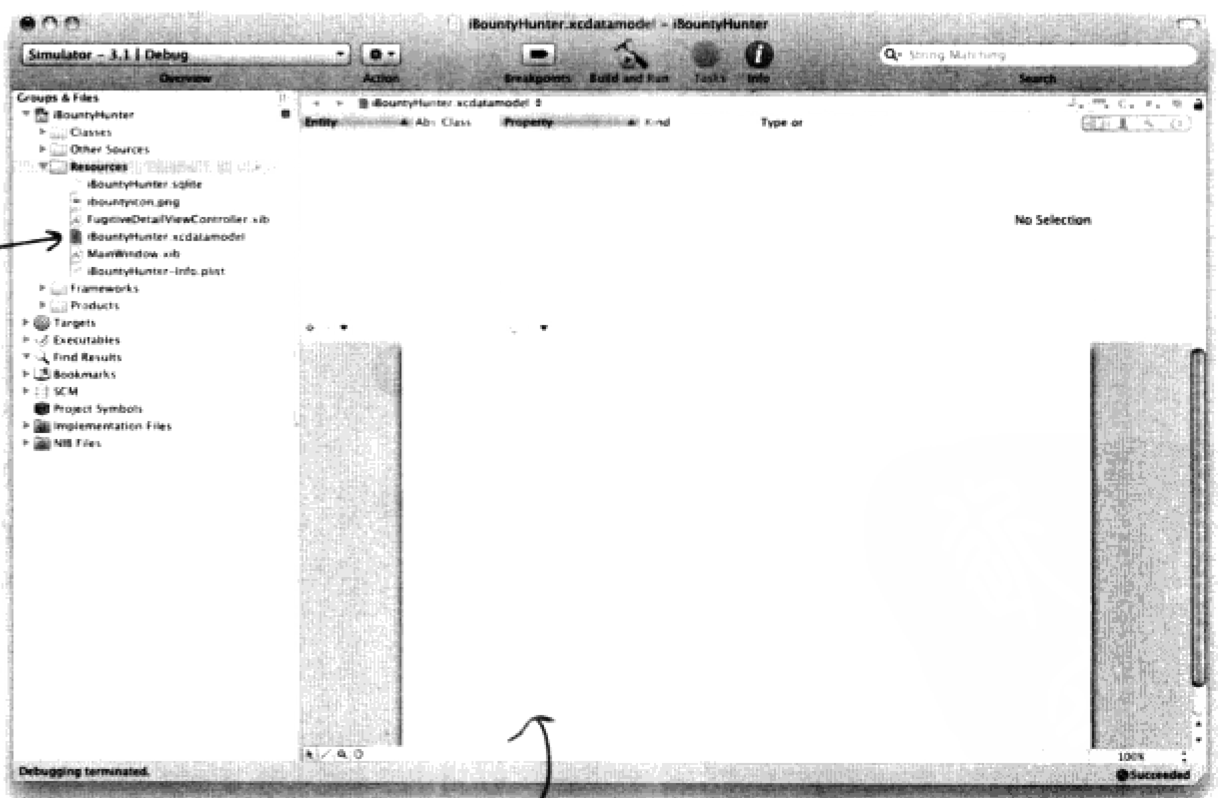
受Core Data控制的实体被称为受控对象。你向Core Data对实体所做的描述（属性、关系、类型信息等），就是通过一个受控对象模型完成的。Core Data会在运行时查看这个受控对象模型，找出如何从它的持久化存储（比如一个数据库）中加载和保存数据。我们使用的Xcode模板自带一个空白的受控对象模型，让我们能以此为基础而上手。



受控对象模型描述了我们要请求或者保存的对象。

它也包含了Core Data从存储中读写这些数据所需的全部信息。

我们的模板自带了一个空白受控对象模型，位于Resources分组中，名为iBountyHunter.xcdatamodel。单击该模型就会出现下面的视图。



模板的设置是：Core Data将在启动时试图加载应用程序中定义的所有受控对象模型。我们则只需要这一个模型。

默认情况下，我们的对象模型是空的；我们需要定义逃犯实体。

技术上你可以在代码中手工创建受控对象模型，不过Xcode的工具令此过程变得容易了许多…

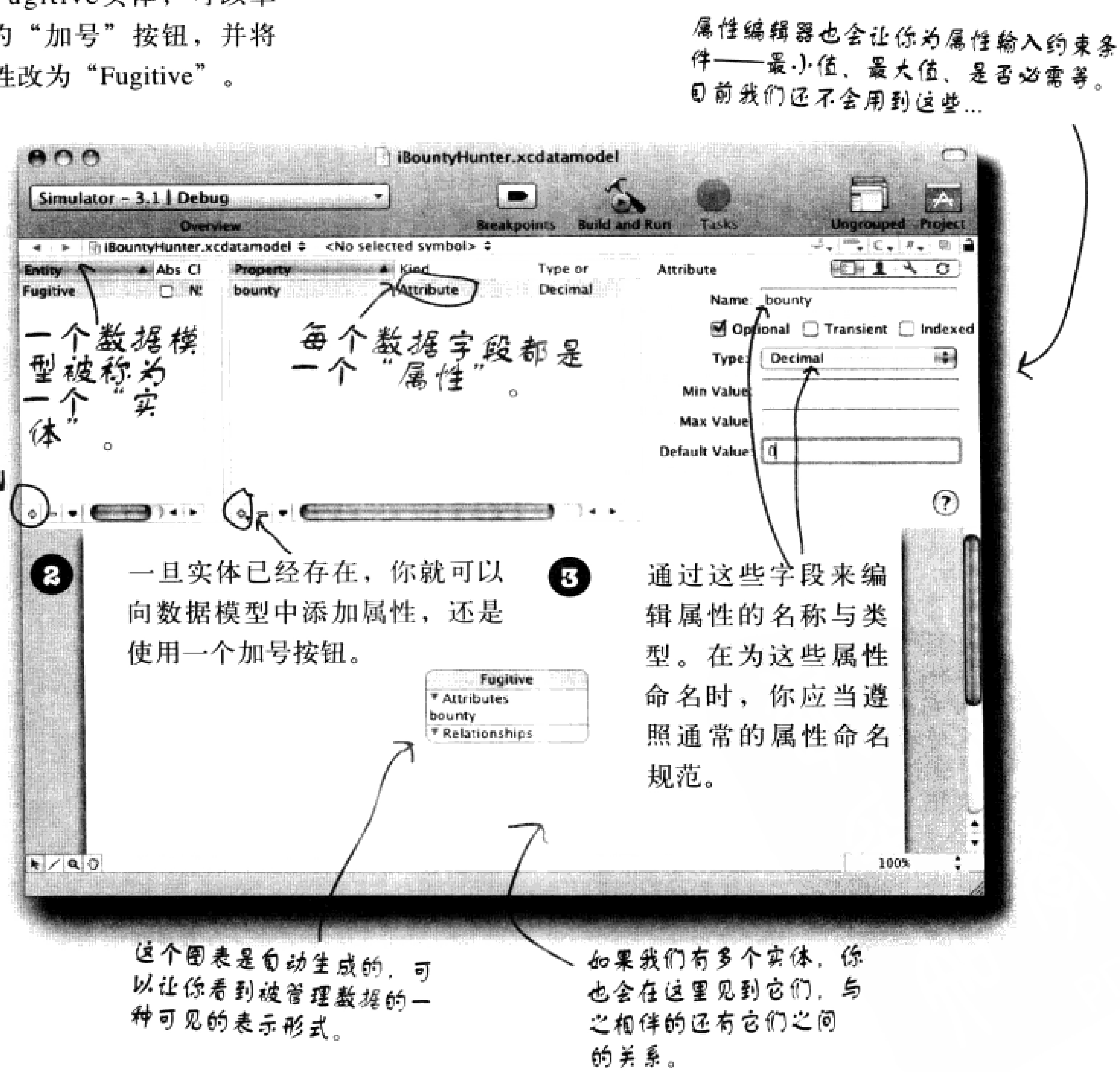
让我们继续创建逃犯实体……

【拼吾爱-最新编程资源的分享下载站】

打造你的Fugitive实体

我们需要在受控对象模型中创建一个逃犯实体。由于我们的Fugitive类与其他类没有任何关系，因此仅需要添加一些属性即可。打开Resources分组中的`iBountyHunter.xcdatamodel`，创建Fugitive数据类型。

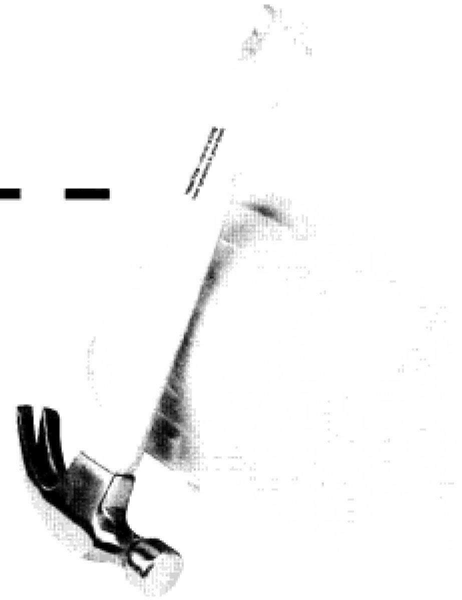
1 要添加Fugitive实体，可以单击此处的“加号”按钮，并将name属性改为“Fugitive”。



2 一旦实体已经存在，你就可以向数据模型中添加属性，还是使用一个加号按钮。

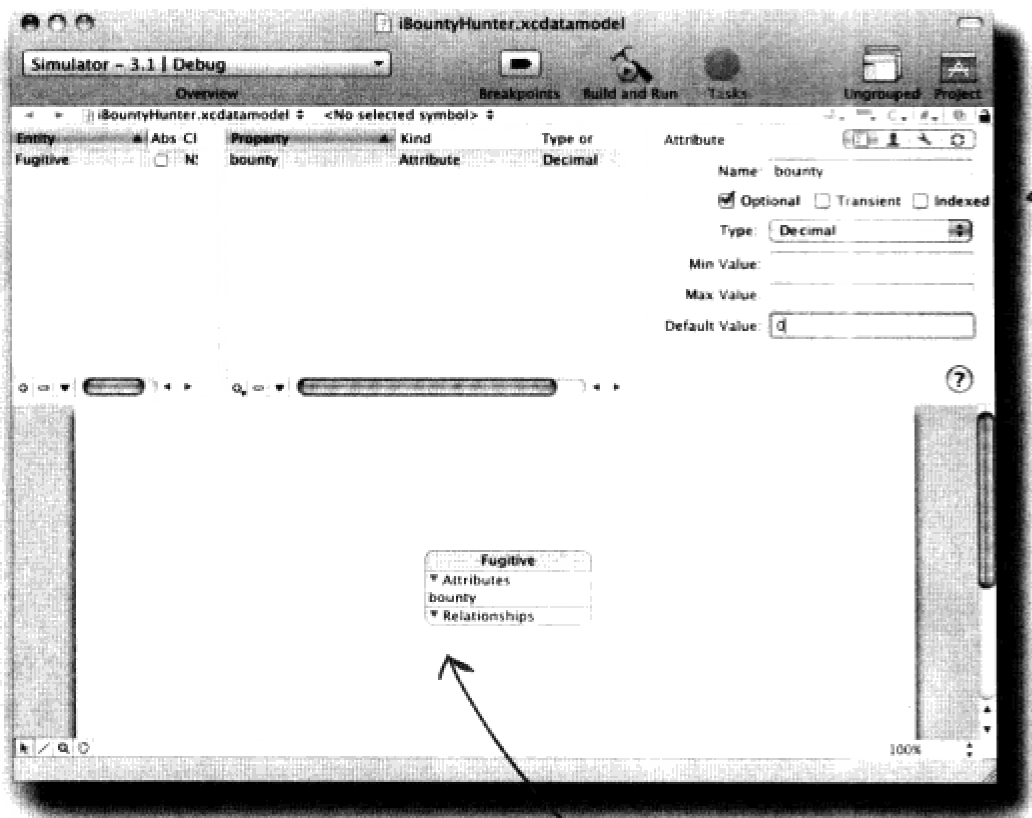
3 通过这些字段来编辑属性的名称与类型。在为这些属性命名时，你应当遵照通常的属性命名规范。

【拼吾爱-最新编程资源的分享下载站】

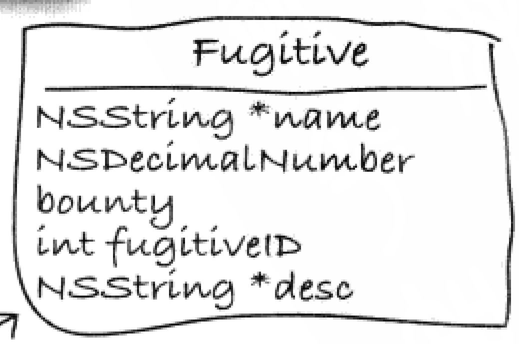


构造受控对象模型

根据我们要储存的逃犯信息，在受控对象模型中完成对Fugitive实体的构造。记住，Core Data类型与我们的Objective-C类型并不完全匹配。一定要确保对属性的命名与下面所示的Fugitive图表中的完全一样。



你应当为自己添加的每个属性都取消选中“Optional”选项—我们想要它们全都是必需的。



你要确保使用的属性名称与我们的完全一样。

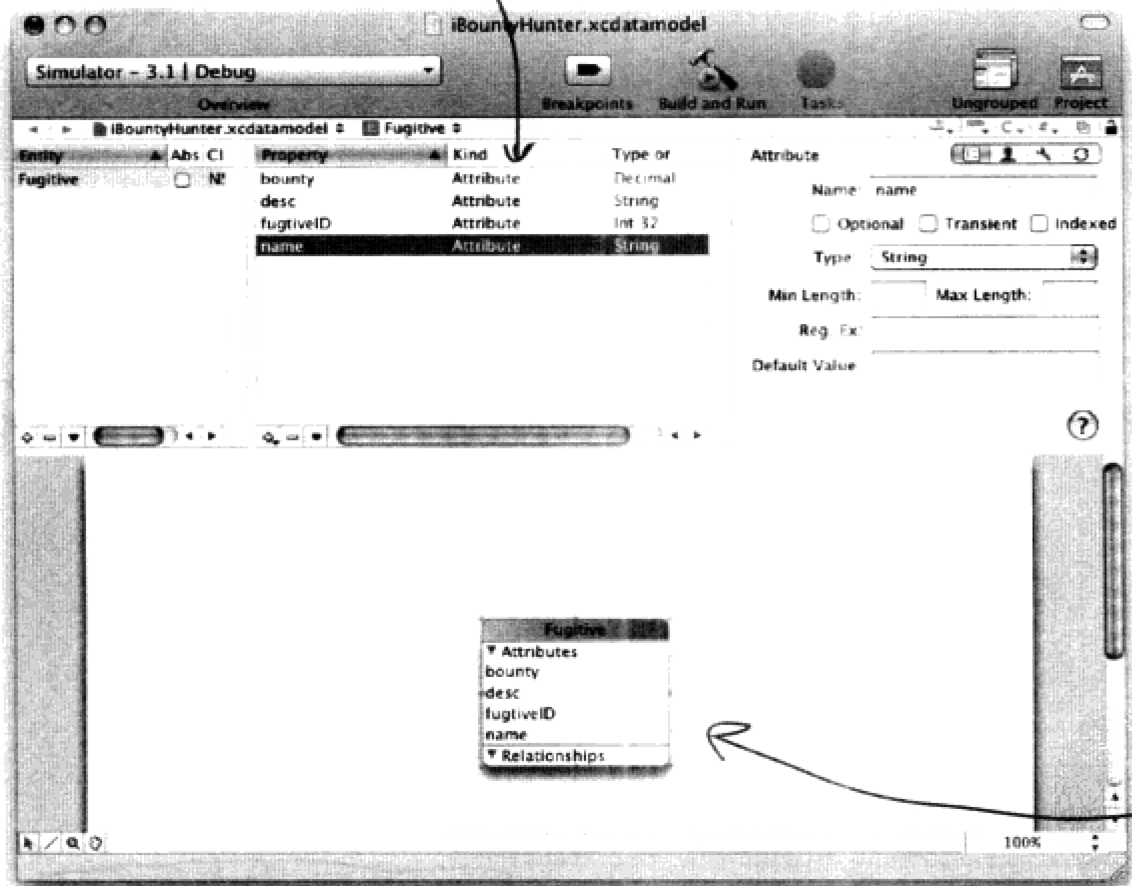
这些是我们使用的Objective-C类型，你需要在构建实体的时候选取相应的Core Data类型。

【拼吾爱-最新编程资源的分享下载站】

构造受控对象模型答案



根据我们要储存的逃犯信息，在受控对象模型中完成对Fugitive实体的构造。记住，Core Data类型与我们的Objective-C类型并不完全匹配。一定要确保对属性的命名与Fugitive图表中的完全一样。





一定要保证你的对象模型与我们的完全一致！

当你编写自己的应用时，有许多种方式来建造数据模型，但是因为我们会给你一个用在iBountyHunter中的数据库，所以你的模型必须与我们的完全一致！

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：为什么你对赏金字段用了一个NSDecimalNumber，而不是一个浮点数或者双精度浮点数呢？

答：我们打算在赏金字段中保存一个货币值，所以要让这个数字的十进制部分保证精确。浮点数和双精度浮点数都是近似的，所以用它们进行货币计算时，常常会在结果应该是$10.00的时候得到$9.999999998。我们之所以为赏金选择了NSDecimalNumber，与Core Data完全无关，只是由我们要保存的内容决定的。

问：在创建属性的时候，Xcode里面出现的标有Transient（临时）和Indexed（索引）的点选框是做什么用的？

答：Transient（临时）点选框表示Core Data不需要保存和加载这个属性。临时属性保存的那些值，通常是因为性能或者便捷方面的需要而计算仅仅一次，而且是可以根据实体中保存的其他数据计算出来的。如果你使用了临时属性，通常会实现一个名为awakeFromFetch:的方法，这个方法会在Core Data加载你的实体后马上被调用。在这个方法中你可以计算出临时属性的值并设置这些属性。

Indexed（索引）点选框告诉Core Data应当尝试在此属性上创建一个索引。Core Data可以利用索引来加速对条目的查询，所以如果你要用某些属性来查找实体（比如客户ID、账户号码等），可以请求Core Data对这些属性进行索引以加速搜索。索引会占据空间，也会拖慢在存储中插入新数据的速度，所以只应该在索引确实能改进查询性能的时候才使用它。

问：我见到有一些用k做名字开头声明的常量。它们有什么不同之处吗？

答：没有。这只不过是一种命名规范。C和C++程序员爱用全大写来命名常量，而苹果公司则倾向于使用小写“k”。

问：如果我需要使用一种Core Data不支持的类型该怎么办？

答：最简单的方法很显然就是试着让你的数据使用一种内建数据类型。如果没法做到，你可以创建自定义类型，并实现一些方法来帮助Core Data保存和加载这些值。实在没有办法的话，你还可以将你的数据插入一种二进制类型（二进制数据或者BLOB）当中，再编写一些代码在运行时对其进行编码解码。

问：Core Data还支持其他类型的持久化吗？

答：在iPhone上Core Data支持3种类型的持久化存储：二进制文件、SQLite数据库和内存中持久化。SQLite存储是最有用的一种，我们在iBountyHunter中就在使用这种方式。它也是默认的持久化方式。二进制文件也可以用，因为它们是原子化的，也就是说所有内容要么一起存储成功，要么没有任何内容被存储。这种方式的问题在于，为了实现原子化，iPhone必须在每次有内容变化时就对整个文件进行读写。在iPhone上这种方法用的不太多。内存中持久化这种类型的存储实际上从来不会被保存到磁盘上，但是让你可以在保存于内存中的数据上使用Core Data提供的所有查询、排序和撤销重做等功能。

问：Core Data在向一个SQLite数据库中写入的时候，都用到了哪些SQL数据类型/表结构？

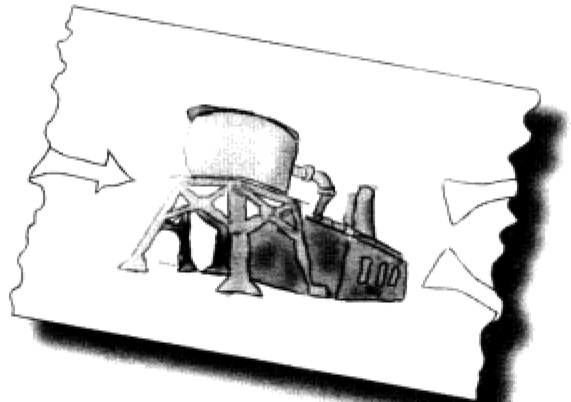
答：简单来说你不用管这些事。虽然它在向SQLite数据库写入，但其格式、类型和结构都不是公开API的一部分，有可能被苹果公司改变。你应该将SQLite数据库作为一个黑盒来对待，只能通过Core Data来访问它。

问：这确实是一个不错的GUI，但仅此而已，我还没看出这和字典比起来有哪些优势之处。看起来还是得做许多工作。

答：我们必须告诉Core Data我们使用的信息是什么样的。现在我们已经完成了这些工作，可以开始将其加入到实际工作中了。

【拼吾爱-最新编程资源的分享下载站】

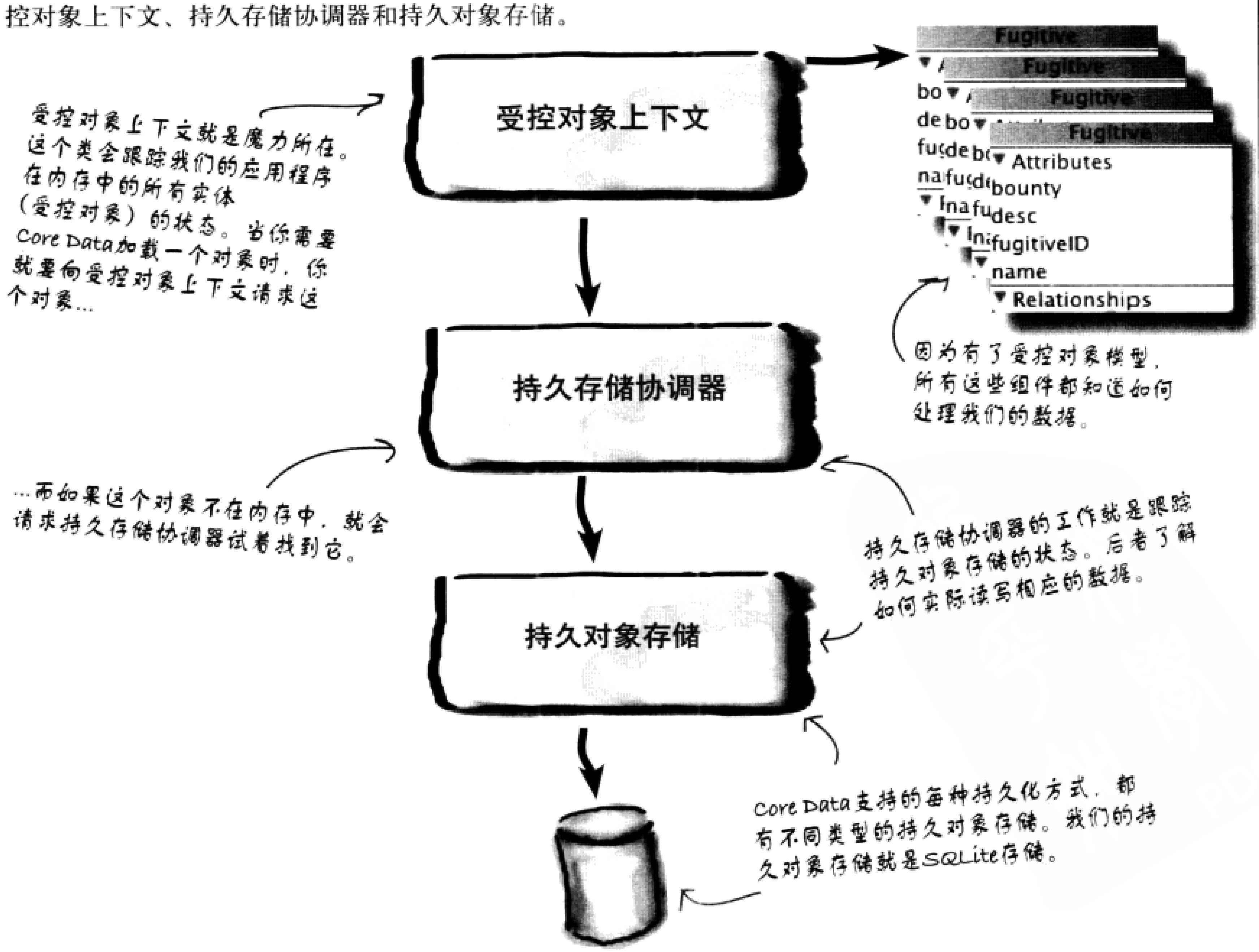
Core Data全接触



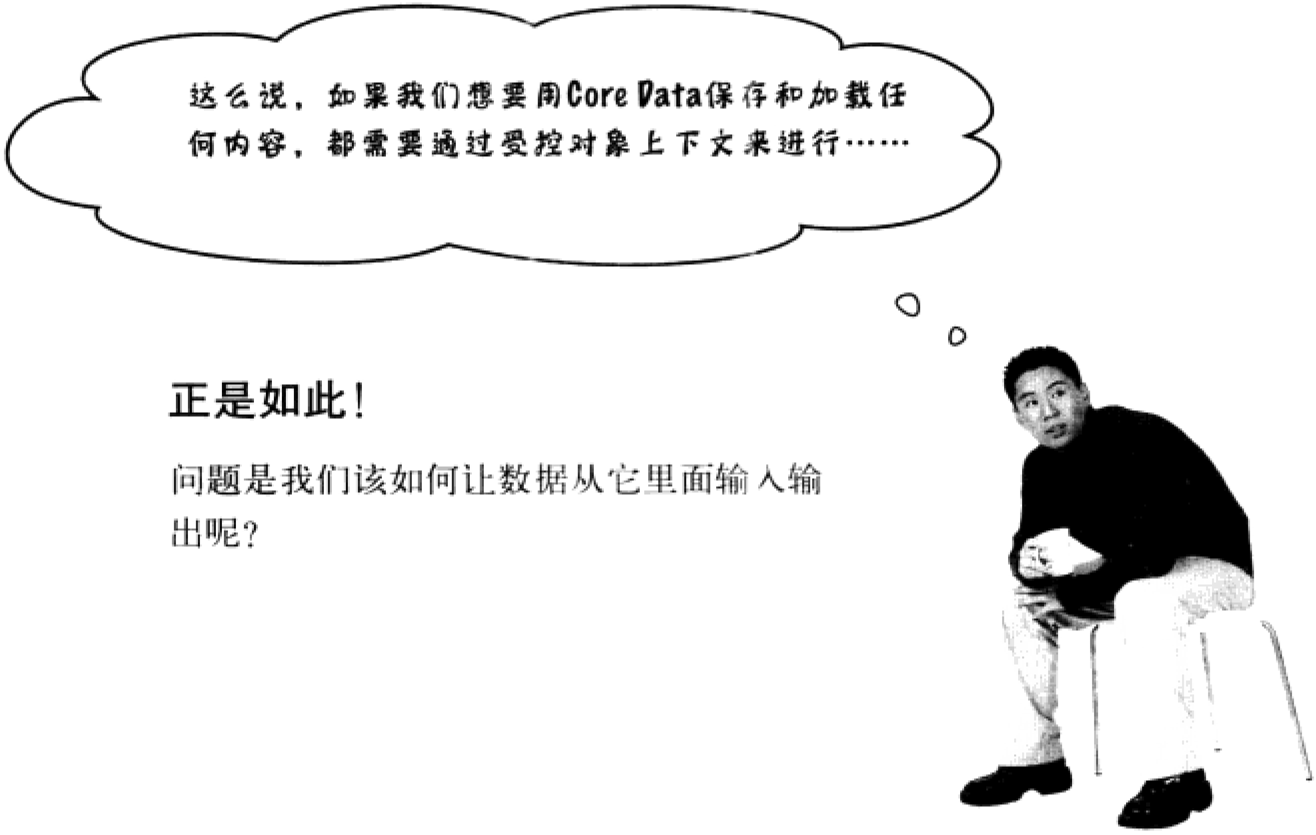
Core Data就是用来管理对象的

到目前为止我们已经谈到了如何向Core Data描述我们的对象，但是还没说如何用它们做实际的工作。要想把这些真正应用起来，我们需要对Core Data的内部做一个快速的了解。

在Core Data内部有一个堆栈，里面有3种关键成员：受控对象上下文、持久存储协调器和持久对象存储。



【拼吾爱-最新编程资源的分享下载站】



正是如此！

问题是我们该如何让数据从它里面输入输出呢？

练习

我们使用的Xcode模板为我们搭建好了Core Data堆栈，但还是得想办法找出如何与受控对象上下文通信。根据到目前为止你了解到的关于Core Data的知识，该如何请求这个框架为你保存和加载数据呢？

- ☐ 使用SQLite命令
- ☐ 编写自定义的保存和加载代码来更新数据
- ☐ 用Core Data生成的类替你完成这项工作

【拼吾爱-最新编程资源的分享下载站】

练习答案

我们使用的Xcode模板为我们搭建好了Core Data堆栈，但还是得想办法找出如何与受控对象上下文通信。根据到目前为止你了解到的关于Core Data的知识，该如何请求这个框架为你保存和加载数据呢？

☐ 使用SQLite命令

我们使用了SQLite存储，但Core Data也支持其他类型的存储方式。它如何使用SQLite对你是完全隐藏的。试图直接用SQL进行访问将是危险的。

☐ 编写自定义的保存和加载代码来更新数据update the data

这样做有两个问题：首先，你仍然不知道数据实际是如何存储的（甚至不知道所用的存储类型）；其次，我们使用Core Data的最大原因就是要避免编写这类代码。

☑ 用Core Data生成的类替你完成这项工作

这正是我们的追求！有了我们的受控对象模型，Core Data了解所需的一切内容，可以为我们创建类并完成所有保存和加载工作，只要对它发出请求即可。

关键要点

- Core Data是一个持久化框架，提供了保存、加载、版本控制以及撤销－重做等功能。
- Core Data可以构建在SQLite数据库之上，或者是二进制文件、临时内存中。
- 受控对象模型定义了实体。我们将请求Core Data使用这些模型进行工作。
- 受控对象上下文是我们访问数据的入口点。它会跟踪活动的受控对象的状态。
- 持久对象存储是Core Data堆栈的一部分，负责处理读写我们的数据。

【拼吾爱-最新编程资源的分享下载站】

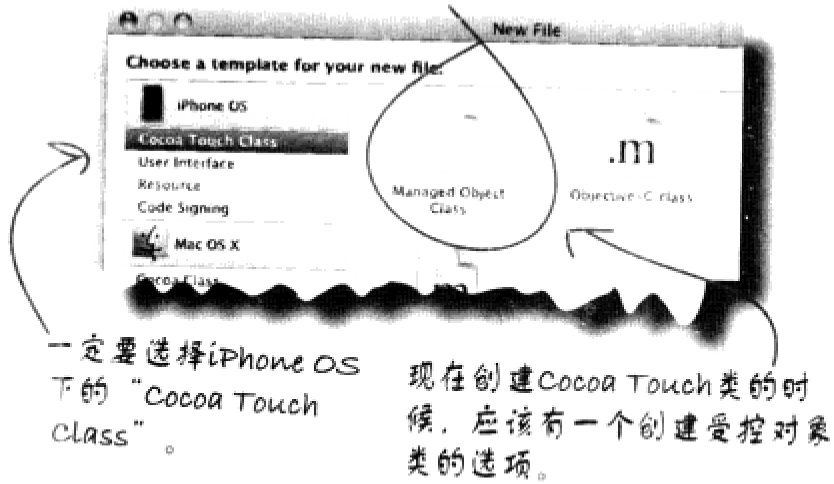
不写一行代码，快速打造Fugitive类

Xcode可以根据我们的受控对象模型创建一个Fugitive类，这个类可以像任何其他类一样被我们使用。



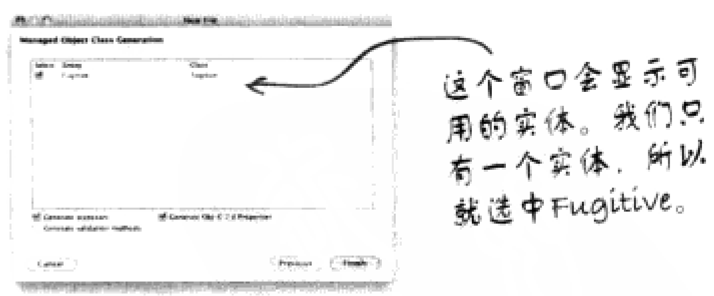
1 **选中iBountyHunter.xcdatamodel并在Fugitive实体上单击**

你需要选中一个Core Data实体，然后才能让Xcode为你生成类。



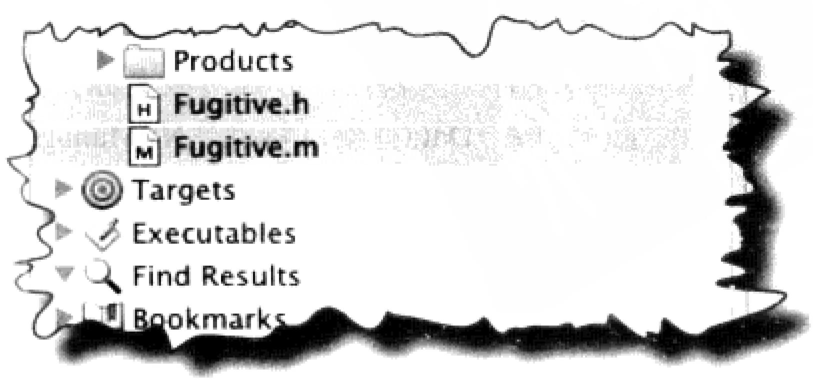
2 **创建一个新的受控对象类...**

在菜单中选择**File→New File**... 会出现一种新类型的文件可以添加到项目中，就是**Managed Object Class**。选择这个文件然后点击**Next**。



3 **...基于Fugitive实体**

你会被问到要使用哪个实体来创建类，你应该选Fugitive。然后单击**Finish**。



4 **就会生成.h和.m文件**

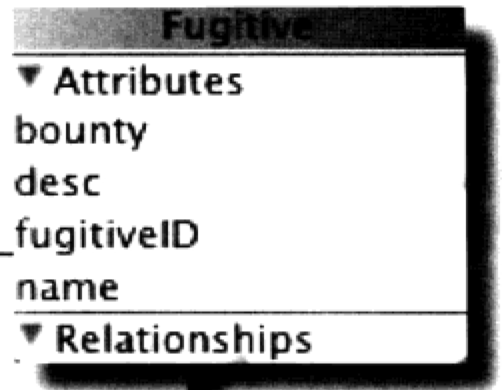
单击**Finish**后，一个Fugitive.h和一个Fugitive.m就会被添加到你的项目中。下一步就是把它们拖拽到Classes分组中。

【拼吾爱-最新编程资源的分享下载站】

我们生成的Fugitive类与我们的受控对象模型吻合

Xcode根据我们的Fugitive实体生成两个新文件：一个头文件Fugitive.h和一个实现文件Fugitive.m。让我们打开两个文件，看一看生成的到底是什么。

新的Fugitive类继承自NSManagedObject—这是一个受控对象。

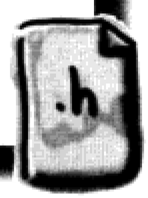



```
#import <CoreData/CoreData.h>

@interface Fugitive :  NSManagedObject
{
}


@property (nonatomic, retain) NSDecimalNumber * bounty;
@property (nonatomic, retain) NSString * name;
@property (nonatomic, retain) NSString * desc;
@property (nonatomic, retain) NSNumber * fugitiveID;

@end
```



Fugitive.h

类拥有我们期望的属性，但是没有成员变量？！？！

我们在受控对象模型里选择的Core Data类型被映射到相应的Objective-C类型。

NSManagedObject会处理所生成属性的存储和内存管理

生成的Fugitive类有一些属性，分别对应名称、描述信息等，但是类里面没有任何成员变量。Core Data框架（特别是NSManagedObject）负责处理这些属性相关的内存管理工作。如果你愿意，也可以改写这一行为，不过大多数情况下它们的行为正是你所需要的。

Fugitive.m里面的内容就更有趣了……

【拼吾爱-最新编程资源的分享下载站】




```
#import "Fugitive.h"
@implementation Fugitive

@dynamic bounty;
@dynamic name;
@dynamic desc;
@dynamic fugtiveID;

@end
```

Fugitive.m

Fugitive类的实现几乎完全是空白！

对！Core Data框架会搞定它。

Fugitive.m中的类几乎是全空的，而且并没有合成那些属性，而是用了一个新的预处理指令@dynamic来声明属性。

NSManagedObject也实现了这些属性

新的@dynamic预处理指令告诉编译器不要操心属性必需的读写方法。不过需要在其他某个地方来提供这些方法，否则在运行时如果有人试图访问这些属性，代码就会崩溃。这里NSManagedObject就再次涉入了。因为NSManagedObject负责为后台的支持属性的成员变量处理内存，它还为读写方法提供运行时实现。依靠NSManagedObject实现的这些方法，你可以得到许多其他简洁的好处：

- NSManagedObject了解何时属性被改变，可以验证新数据，并可以在发生改变时通知其他类。
- NSManagedObject可以延迟获取属性的信息，直到有人对其进行请求为止。例如，与其他对象的关系就会这样处理。
- NSManagedObject可以跟踪属性的变化，并提供撤销－重做支持。

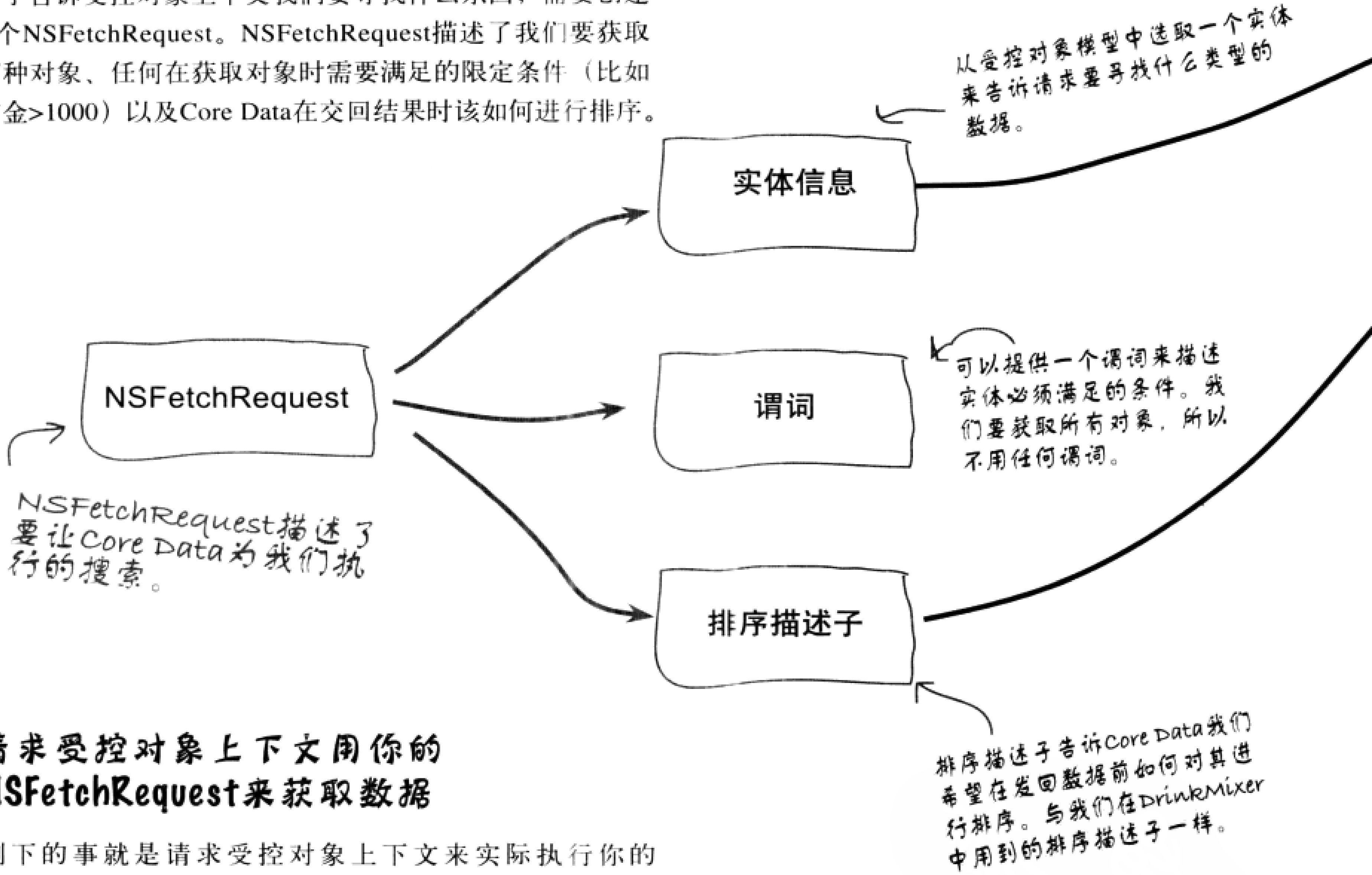
所有这些功能，你不用写一行代码就能获得。

现在只要请求Core Data加载一个Fugitive对象就行了……

【拼吾爱-最新编程资源的分享下载站】

用一个NSFetchRequest描述你的搜索

为了告诉受控对象上下文我们要寻找什么东西，需要创建一个NSFetchRequest。NSFetchRequest描述了我们要获取何种对象、任何在获取对象时需要满足的限定条件（比如赏金>1000）以及Core Data在交回结果时该如何进行排序。



请求受控对象上下文用你的NSFetchRequest来获取数据

剩下的事就是请求受控对象上下文来实际执行你的NSFetchRequest了。这意味着我们需要一个对受控对象上下文的引用。幸运的是，模板已经在应用代理中为我们设置好了。通过如下方法，我们可以得到它：

```
iBountyHunterAppDelegate *appDelegate =
    (iBountyHunterAppDelegate*)[[UIApplication sharedApplication] delegate];
NSManagedObjectContext *managedObjectContext =
    appDelegate.managedObjectContext;
```

【拼吾爱-最新编程资源的分享下载站】


```
NSFetchRequest *request = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription
entityForName:@"Fugitive" inManagedObjectContext:managedObjectContext];
[request setEntity:entity];

NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
initWithKey:@"name" ascending:YES];
NSArray *sortDescriptors = [[NSArray alloc]
initWithObjects:sortDescriptor, nil];
[request setSortDescriptors:sortDescriptors];
[sortDescriptors release];
[sortDescriptor release];

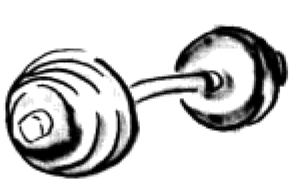

NSError *error;
NSMutableArray *mutableFetchResults = [[managedObjectContext
executeFetchRequest:request error:&error] mutableCopy];
if (mutableFetchResults == nil) {
    // 可能会需要做些更有意义的事...
    NSLog(@"Can't load the Fugitive data!");
}

[mutableFetchResults release];
[request release];
```

我们通过名称来指定实体是Fugitive。

我们希望逃犯按照名字字母顺序排序。

剩下的事就是让受控对象上下文去执行我们的数据获取请求了。我们请求它以数组形式交回结果，然后清除我们的引用。



脑力举重

现在，我们要把这段代码放在哪里呢？而我们又要把结果存储于何处呢？到底要怎样显示获取到的数据呢？

【拼吾爱-最新编程资源的分享下载站】

脑力举重 答案

现在，我们要把这段代码放在哪里呢？而我们又要把结果存储于何处呢？到底要怎样显示获取到的数据呢？

由于鲍勃想要在视图出现时立刻看到列表，获取对象的代码需要放在FugitiveViewController.m文件的viewWillAppear方法中。

对于如何存储结果，我们得到的是一个数组，但是我们随即就将其释放了。我们需要在视图控制器中保留一个对此数组的引用。

为了实际显示这组数据，我们需要实现cellForRowAtIndexPath方法，从数组中抽取数据。

拿起笔来

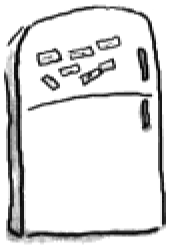
让我们把这些片断全都组装到应用中。

1. **创建一个可变数组来保存获取到的条目。**
在FugitiveViewController中创建一个数组，命名为items，用来存放获取的结果。不要忘记合成属性与清理内存。

2. **在FugitiveViewController.m中引入适当的头文件。**
你要确保在FugitiveListViewController.m中为应用代理和Fugitive类#import了头文件。

3. **在viewWillAppear中实现数据获取代码。**
用我们在前几页中讲过的内容，让数据获取能够正常工作起来。你需要从代理处获得受控对象上下文的引用，创建数据获取请求，然后执行请求。记得要修改代码，把结果赋给我们刚刚创建的数组，以便保持住它们。

【拼吾爱-最新编程资源的分享下载站】



表格单元魔力贴

用下面的代码片断为逃犯列表来定制表格单元。

```
________________________________________

________________________________________ (UITableView *)tableView {
        return 1;
}

// Customize the number of rows in the table view.
 - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section {
 ____________________
}

// Customize the appearance of table view cells.
 - (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath {
 ______________________

        UITableViewCell *cell = ________________________________________
CellIdentifier];
        if (cell == nil) {
              cell = [[[UITableViewCell alloc] initWithStyle:
                                                              autorelease];
         ______________________________________________
        }
        // Set up the cell...

  ________________________________________
  ________________________________________
  ________________________________________
}
```

```
return [items count];
```

```
return cell;
```

```
= fugitive.name;
```

```
Fugitive *fugitive
```

```
UITableViewCellStyleDefault reuseIdentifier:CellIdentifier]
```

```
tableView dequeueReusableCellWithIdentifier:
```

```
cell.textLabel.text
```

```
#pragma mark table view methods
```

```
- (NSInteger) numberOfSectionsInTableView:
```

```
static NSString *CellIdentifier = @"Cell";
```

```
= [items objectAtIndex:indexPath.row];
```

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

要实现的代码有很多，但当你写完以后，Core Data就会按你的需要为逃犯列表去获取数据。

1 创建一个可变数组来保存获取到的条目。

```
#import <UIKit/UIKit.h>
@interface FugitiveListViewController : UITableViewController {
      NSMutableArray *items;
}
@property(nonatomic, retain) NSMutableArray *items;
@end
```

FugitiveListViewController.h

2 在FugitiveViewController.m中引入适当的头文件。

```
#import "FugitiveListViewController.h"
#import "iBountyHunterAppDelegate.h"
#import "Fugitive.h"
@implementation FugitiveListViewController
@synthesize items;
```

```
- (void)dealloc {
    [items release];
    [super dealloc];
}
```

FugitiveListViewController.m

【拼吾爱-最新编程资源的分享下载站】

❸ 在viewWillAppear中实现数据获取代码。

```
- (void) viewWillAppear:(BOOL)animated {
      [super viewWillAppear:animated];
      iBountyHunterAppDelegate *appDelegate =
(iBountyHunterAppDelegate*)[[UIApplication sharedApplication] delegate];
      NSManagedObjectContext *managedObjectContext = appDelegate.
managedObjectContext;
      NSFetchRequest *request = [[NSFetchRequest alloc] init];
      NSEntityDescription *entity = [NSEntityDescription
entityForName:@”Fugitive” inManagedObjectContext:managedObjectContext];
      [request setEntity:entity];

      NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
initWithKey:@”name” ascending:YES];
      NSArray *sortDescriptors = [[NSArray alloc]
initWithObjects:sortDescriptor, nil];
      [request setSortDescriptors:sortDescriptors];
      [sortDescriptors release];
      [sortDescriptor release];

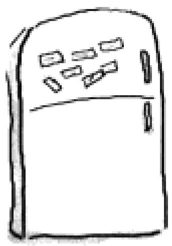
      NSError *error;
      NSMutableArray *mutableFetchResults = [[managedObjectContext
executeFetchRequest:request error:&error] mutableCopy];
      if (mutableFetchResults == nil) {
            // 进行错误处理。
      }

      self.items = mutableFetchResults;
      [mutableFetchResults release];
      [request release];
}
```



FugitiveListViewController.m

【拼吾爱-最新编程资源的分享下载站】



表格单元魔力贴答案

用下面的代码片断为逃犯列表来定制表格单元。

```
#pragma mark table view methods

- (NSInteger) numberOfSectionsInTableView: (UITableView *)tableView {
    return 1;
}


// Customize the number of rows in the table view.
 - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section {
        return [items count];
}


// Customize the appearance of table view cells.
 - (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath {
        static NSString *CellIdentifier = @"Cell";

        UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
CellIdentifier];
        if (cell == nil) {
            cell = [[[UITableViewCell alloc] initWithStyle:
UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
        }

        // Set up the cell...
        Fugitive *fugitive = [items objectAtIndex:indexPath.row];
        cell.textLabel.text = fugitive.name;
        return cell;
}
```

这就是Core Data的工作方式。数据被存储在正常的Objective-C对象中。

按图索骥！ 要完全打造好你的表格视图，请确保在Interface Builder中，逃犯列表表格视图的数据源被设置为FugtiveListViewController。

【拼吾爱-最新编程资源的分享下载站】

对号入座

将每个Core Data概念与它所做的工作匹配起来。

受控对象模型

NSManagedObject

受控对象上下文

NSFetchRequest

NSSortDescriptor

描述了你要在数据上执行的搜索。包括了要取得的类型信息、数据必须满足的条件以及结果应如何排序。

负责跟踪应用程序中活动的受控对象的状态。你的所有获取与保存数据的请求都要通过它来完成。

以一种通用的方式来确定数据应如何排序。你可以指定数据排序所依据的字段，以及排序方式。

描述应用程序中的实体，包括类型信息、数据约束以及实体之间的关系。

Core Data实体的一个Objective-C版本。它的子类代表了你要通过Core Data来保存和加载的数据。提供了对监视数据改变、延迟加载以及数据校验等功能的支持。

【拼吾爱-最新编程资源的分享下载站】

对号入座答案

将每个Core Data概念与它所做的工作匹配起来。

受控对象模型	描述了你要在数据上执行的搜索。包括了要取得的类型信息、数据必须满足的条件以及结果应如何排序。
NSManagedObject	负责跟踪应用程序中活动的受控对象的状态。你的所有获取与保存数据的请求都要通过它来完成。
受控对象上下文	以一种通用的方式来确定数据应如何排序。你可以指定数据排序所依据的字段，以及排序方式。
NSFetchRequest	描述应用程序中的实体，包括类型信息、数据约束以及实体之间的关系。
NSSortDescriptor	Core Data实体的一个Objective-C版本。它的子类代表了你要通过Core Data来保存和加载的数据。提供了对监视数据改变、延迟加载以及数据校验等功能的支持。

【拼吾爱-最新编程资源的分享下载站】



你需要下载逃犯列表的拷贝。

浏览网址http://www.headfirstlabs.com/iphonedev，下载iBountyHunter.sqlit。在iBountyHunter项目上单击右键并选择Add→Existing Files...，要确保选中了“将文件拷贝到项目中”。

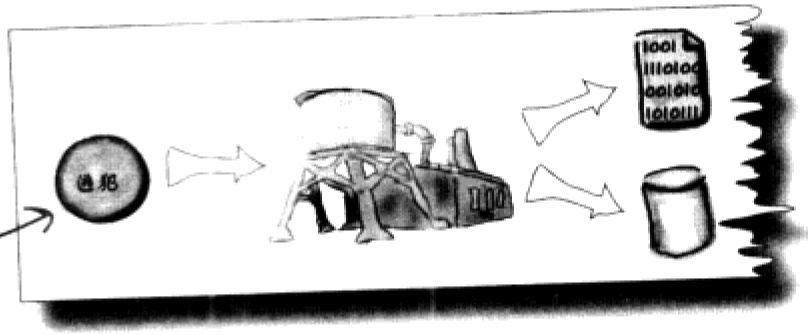
脑力练习

我们怎样才能告诉Core Data从这个文件中加载数据呢？

【拼吾爱-最新编程资源的分享下载站】

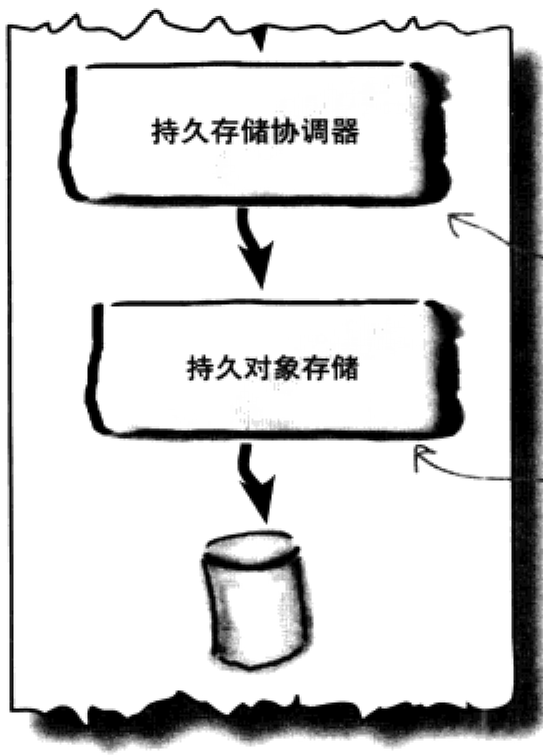
将数据库作为资源添加进来

我们已经将所有加载数据的代码安排就位——这是Core Data模板自带的。但我们该如何令其去实际加载数据库呢？



回到Core Data堆栈

记得吗，之前我们提到过的Core Data堆栈？我们已经令受控对象上下文一切就位，现在我们开始对数据的实际来源感兴趣了。就像受控对象上下文一样，模板也为我们设置好了堆栈剩下的部分。



让我们看一下应用代理中的模板代码……

【拼吾爱-最新编程资源的分享下载站】

模板设置为使用SQLite数据库

Core Data模板对持久存储协调器的设置是：使用一个名字与我们项目相同的SQLite数据库。只要数据库的名字是iBountyHunter.sqlite，那么Core Data就可以正常运转。

```
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {
      if (persistentStoreCoordinator != nil) {
       return persistentStoreCoordinator;
    }

    NSURL *storeUrl = [NSURL fileURLWithPath: [[self
applicationDocumentsDirectory] stringByAppendingPathComponent:
@"iBountyHunter.sqlite"]];
      NSError *error = nil;
    persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] i
nitWithManagedObjectModel:[self managedObjectModel]];

    if (![persistentStoreCoordinator addPersistentStoreWithType:NSSQLite
StoreType configuration:nil URL:storeUrl options:nil error:&error]) {
  NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
 abort();
    }
 return persistentStoreCoordinator;
}
```

模板设置为使用与项目同名的数据库。

模板代码将一个持久对象存储添加到协调器中，协调器类型被配置为NSSQLiteStoreType。


iBountyHunterAppDelegate.m

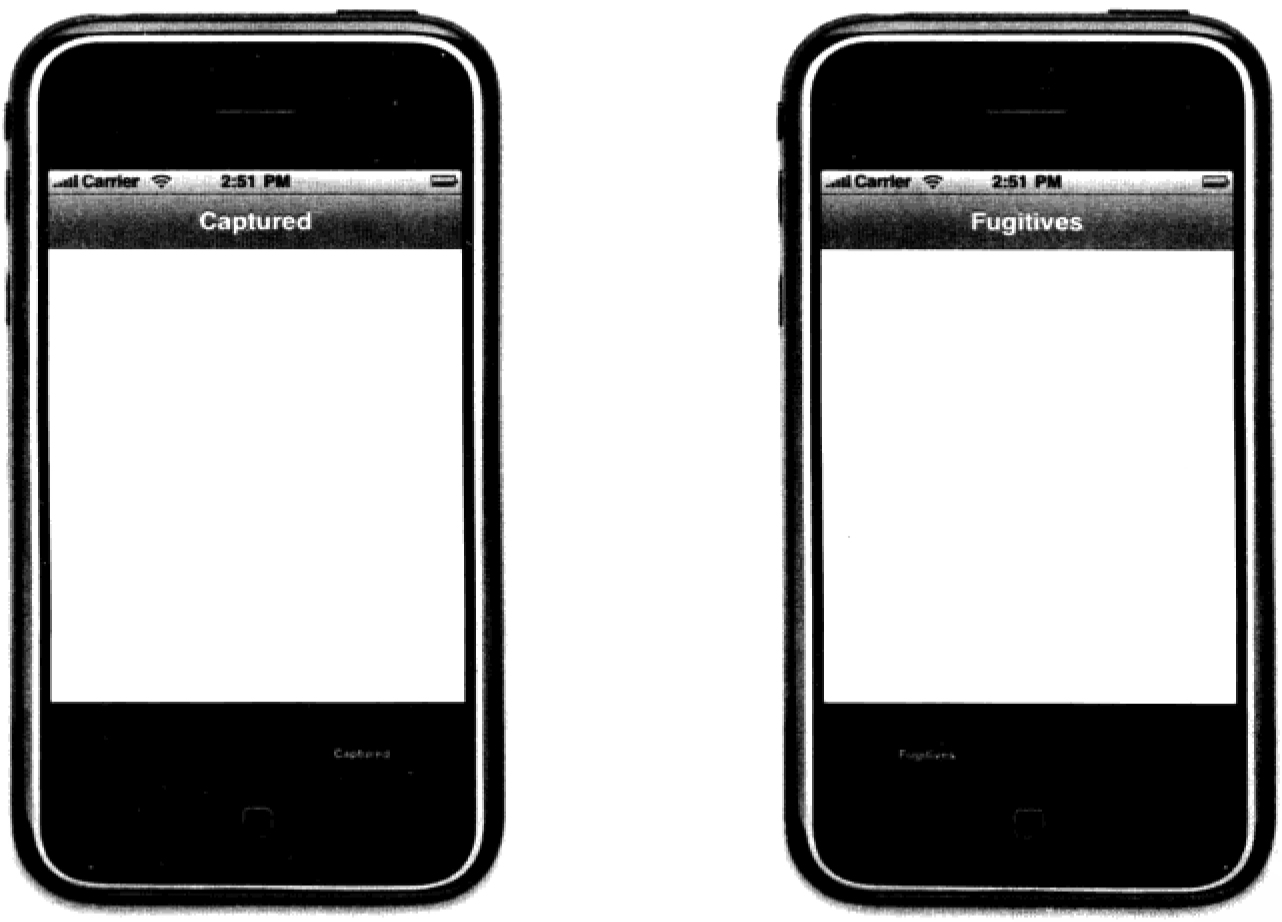


运行测试

现在数据已经就位，而持久对象存储也可以原样不动直接使用，请继续进行下一步，运行应用。

【拼吾爱-最新编程资源的分享下载站】

运行测试



数据在哪里？

【拼吾爱-最新编程资源的分享下载站】



我们把数据库添加到项目中了。代码看起来也没问题。这样做在DrinkMixer里工作得挺好。这是怎么回事？

Core Data关注的是另外一些地方。

我们的问题在于Core Data寻找数据库的方式。哎呀，问题实际上要比这个稍微复杂一点点。

留神！

这段代码只能在模拟器中工作!!

这段用来保存plist的代码在模拟器上工作状况良好，但是在真实设备上就悲惨地失败了。问题在于文件权限以及应用允许保存数据的位置。我们会在第7章中进一步讲解这个问题，现在先用着这个版本。这是一个很好的例子，展示了能够运行在模拟器上，却在真实设备上行为有差异的情况。

iPhone应用是只读的

回头看看DrinkMixer，我们使用应用程序束从一个plist中加载应用程序的数据。这种方法工作良好，我们的数据毫无困难地被加载。但是记得我们是怎么说的吗？这只能在模拟器中工作。现在该澄清这一问题了。作为iPhone安全措施的一部分，安装在设备上的应用程序是只读的。你可以读取应用程序中所带的任何资源束，但是你无法对其进行修改。Core Data模板假定你会对数据库进行读和写，所以它甚至根本不会去费心查看应用程序束中是否有数据库。

Core Data模板会在应用程序文档目录下寻找数据库，而不是在应用程序束中。

```
NSURL *storeUrl = [NSURL fileURLWithPath: [[self
applicationDocumentsDirectory] stringByAppendingPathComponent:
@"iBountyHunter.sqlite"]];
```



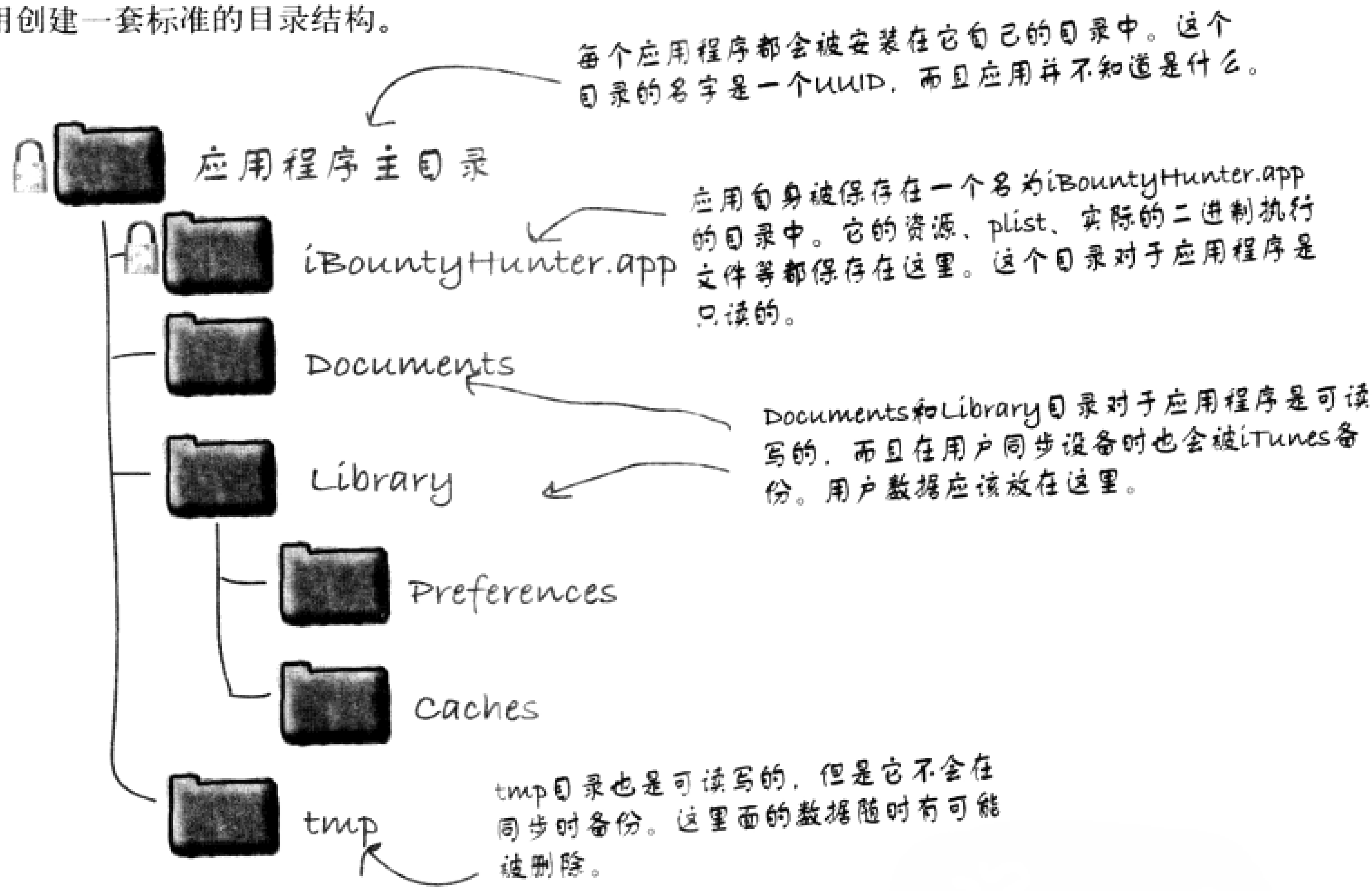
iBountyHunterAppDelegate.m

我们需要更仔细地看一看这些目录是被如何设置的……

【拼吾爱-最新编程资源的分享下载站】

iPhone的应用程序结构决定了你能在哪些地方读和写

出于安全与稳定性的考虑，iPhone OS对文件系统封锁得相当紧密。当安装一个应用程序时，iPhone OS会在设备的/User/Applications目录下用一个唯一标识符创建一个子目录。应用程序就被安装在这个目录中，同时会为应用创建一套标准的目录结构。



使用Documents目录来存储用户数据

由于大多数Core Data应用程序都需要对数据进行读和写，所以模板将Core Data堆栈设置为从Documents目录中读写。一个应用程序可以找出自己的本地目录的位置，方法是使用NSSearchPathForDirectoriesInDomains，就像模板在应用代理中所做的那样：

```
- (NSString *)applicationDocumentsDirectory {
   return [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
        NSUserDomainMask, YES) lastObject];
}
```

【拼吾爱-最新编程资源的分享下载站】

将数据库复制到正确的位置

应用程序第一次启动时，我们需要查看我们的Documents目录中是否有一份数据库的拷贝。如果有的话，我们不会想把它搞乱掉的。如果没有，则需要向这个位置复制一份。

你需要在iBountyHunterAppDelegate.h中声明这个方法。

```
-(void)createEditableCopyOfDatabaseIfNeeded {
    // 首先，测试是否存在——我们可不想把用户的数据库抹掉
    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *documentsDirectory = [self applicationDocumentsDirectory];
    NSString *writableDBPath = [documentsDirectory stringByAppendingPathCompo
nent:@"iBountyHunter.sqlite"];

    BOOL dbexists = [fileManager fileExistsAtPath:writableDBPath];
    if (!dbexists) {
    // 可写数据库不存在，所以将默认库拷贝到正确的位置。

    NSString *defaultDBPath = [[[NSBundle mainBundle] resourcePath] stringByA
ppendingPathComponent:@"iBountyHunter.sqlite"];

    NSError *error;
    BOOL success = [fileManager copyItemAtPath:defaultDBPath
toPath:writableDBPath error:&error];
    if (!success) {
    NSAssert1(0, @"Failed to create writable database file with message
'%@'.", [error localizedDescription]);
      }
    }
}
```

此处我们从应用程序束中抓取主数据库；这是一个只读副本。

将其从只读目录复制到可写目录。

```
- (void)applicationDidFinishLaunching:
(UIApplication *)application {
        [self createEditableCopyOfDatabaseIfNeeded];
}
```

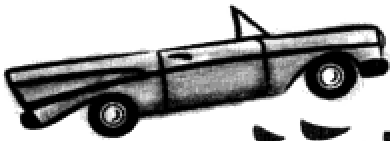


iBountyHunterAppDelegate.m

按图索骥！

现在应用已经知道如何复制数据库了，你需要将应用的旧版本卸载，才能将以前Core Data创建的空数据库删掉。当你再次构建并运行应用之后，我们的新代码就会将正确的数据库复制到位。

【拼吾爱-最新编程资源的分享下载站】



运行测试

现在应用知道去哪儿找数据库了，应该能正确加载数据了。



【拼吾爱-最新编程资源的分享下载站】

有问必答

问：为什么我们在DrinkMixer里用plist的时候就不用管这些目录的问题呢？

答： 我们只是在模拟器中运行了DrinkMixer，而模拟器并不像真实设备一样有强制的目录权限。如果在真正的设备上使用DrinkMixer，我们基本上会遇到一样的问题。之所以对于iBountyHunter这个问题表现得如此明显，是因为Core Data的配置就是要在正确的位置上找到一个可以写入的数据库，这个位置就是应用程序的Documents目录。

问：我怎样才能得到应用程序的其他目录的路径呢？

答： 只要使用NSSearchPathForDirectoriesInDomains但辅以不同的NSSearchPathDirectory常量就可以了。其中的大多数你应该从来也用不到；NSDocumentsDirectory是最常见的。你绝对不应假定自己了解目录结构，或者是知道如何自己找到其他目录；永远都要用API查找你要用到的特定目录。

问：那么当有人卸载我的应用程序的时候，数据会怎么样呢？

答： 当一个应用程序被从设备上删除时，整个应用程序目录都会被删掉，所以不管是数据还是缓存、偏好设置等，都会被删除。

问：关于NSFetchRequest用到的谓词，看起来很重要啊。我们还会进一步讨论这个问题么？

答： 是的！我们会在第8章再来讲这个问题。

问：那么，在一个应用程序中总是只有一个受控对象上下文吗？

答： 不，如果需要可以有多个上下文。对于大多数应用来说，一个就足够了，但是如果你想要将一组编辑控件隔离开，或是将数据从一个数据源迁移到另一个，根据需要也可以创建并配置多个受控对象上下文。

问：我真没看出持久存储协调器有什么用处。它到底干嘛用的？

答： 我们的应用程序只使用了一个持久对象存储，但是 Core Data是支持多个存储的。例如，你可以让一个客户信息来自一个数据库，而产品信息出自另外一个数据库。你可以配置两个独立的持久对象存储，而让持久存储协调器根据所连接的数据库来决定该用哪一个存储。

问：对象模型呢？我们能有多个对象模型吗？

答： 正是——实际上我们正打算在第8章中说说这事儿呢。

问：我必须总是从受控对象上下文中获取我的NSManagedObject吗？如果我要创建一个新的该怎么办？

答： 不用总是这样，新建的必须被添加到上下文中——然而，你不能简单地对它们进行内存分配和初始化。你需要根据它们的实体描述来创建它们，比如：[NSEntityDescription insertNewObjectForEntityForName:@"Fugitive" inManagedObjectContext:managedObjectContext]；这样就可以返回一个新的Fugitive实例，然后你就可以正常使用它了。

【拼吾爱-最新编程资源的分享下载站】



正是。

我们让数据库加载了细节信息，但是用户还无法看到。现在我们只需要打造好细节视图，就可以显示出这些信息了。

iBountyHunter待办事项列表

1. ~~为逃犯与被捕视图创建视图控制器（.h和.m文件）~~。
2. ~~创建标签栏视图，并为之添加标签栏控制器，以及一个对应用代理的引用~~。
3. ~~为逃犯与被捕视图添加导航控制器~~。
4. ~~为逃犯与被捕视图构建表格视图~~。
5. 创建一个细节视图nib，以及视图控制器的.h和.m文件。

你差不多完成列表上的工作了！

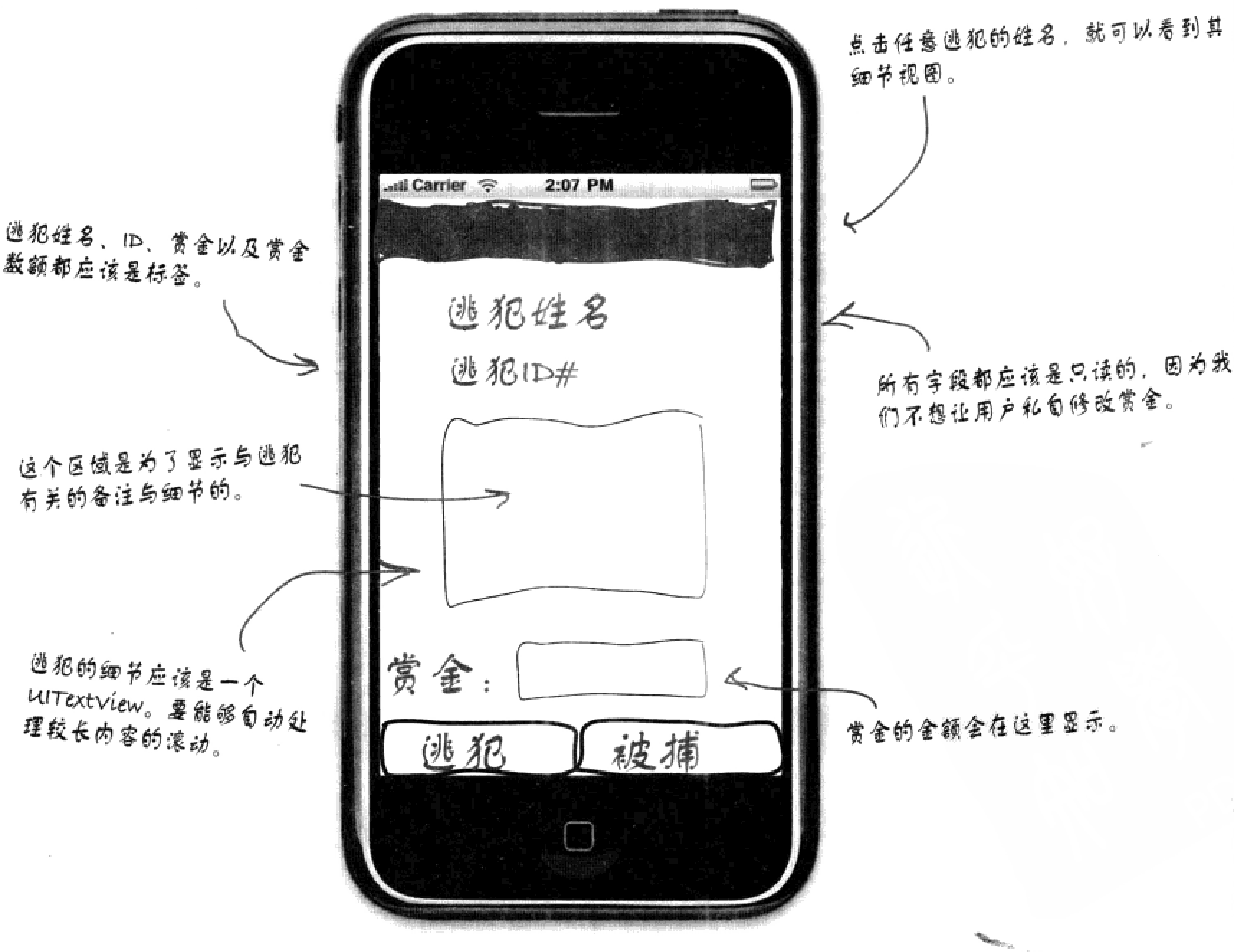
你绝对知道该怎么做这些事……

【拼吾爱-最新编程资源的分享下载站】

长篇练习

打造细节视图对于你来说已不新鲜——那么就开始吧！根据我们之前大致勾画的细节视图，下面就是你要做的工作。

- [ ] 创建一个新的视图控制器和nib，命名为FugitiveDetailViewController。
- [ ] 用Interface Builder为nib进行布局，加入我们需要的控件。
- [ ] 然后修改新的视图控制器，加入我们要进行设置的控件对应的输出口，以及一个对当前正在显示的Fugitive的引用。



【拼吾爱-最新编程资源的分享下载站】

长篇练习答案

遍历代码，并检查输出口、声明和内存释放等。

为添加新视图，你需要以下文件：FugitiveDetailViewController.h、FugitiveDetailViewController.m以及FugitiveDetailViewController.xib.

要创建这些文件，可以从菜单中选择**File → New**，然后选中标有“With XIB for User Interface”的点选框。之后，你还需要在Xcode中将.xib文件移动到**/Resources**下。

当你创建新类的文件时，会有.m、.h和.xib文件。

你要确保将DetailController.xib文件移动到/Resources文件夹下。



【拼吾爱-最新编程资源的分享下载站】


```
@class Fugitive;
@interface FugitiveDetailViewController : UIViewController {
      Fugitive *fugitive;
      UILabel *fugitiveNameLabel;
      UILabel *fugitiveIdLabel;
      UITextView *fugitiveDescriptionView;
      UILabel *fugitiveBountyLabel;
}
@property (nonatomic, retain) Fugitive *fugitive;
@property (nonatomic, retain) IBOutlet UILabel *fugitiveNameLabel;
@property (nonatomic, retain) IBOutlet UILabel *fugitiveIdLabel;
@property (nonatomic, retain) IBOutlet UITextView *fugitiveDescriptionView;
@property (nonatomic, retain) IBOutlet UILabel *fugitiveBountyLabel;
@end
```

FugitiveDetailViewController.h

```
#import "FugitiveDetailViewController.h"
#import "Fugitive.h"
@implementation FugitiveDetailViewController
@synthesize fugitive, fugitiveNameLabel, fugitiveIdLabel,
fugitiveDescriptionView, fugitiveBountyLabel;

- (void)dealloc {
      [fugitive release];
      [fugitiveNameLabel release];
      [fugitiveIdLabel release];
      [fugitiveDescriptionView release];
      [fugitiveBountyLabel release];
    [super dealloc];
}
@end
```

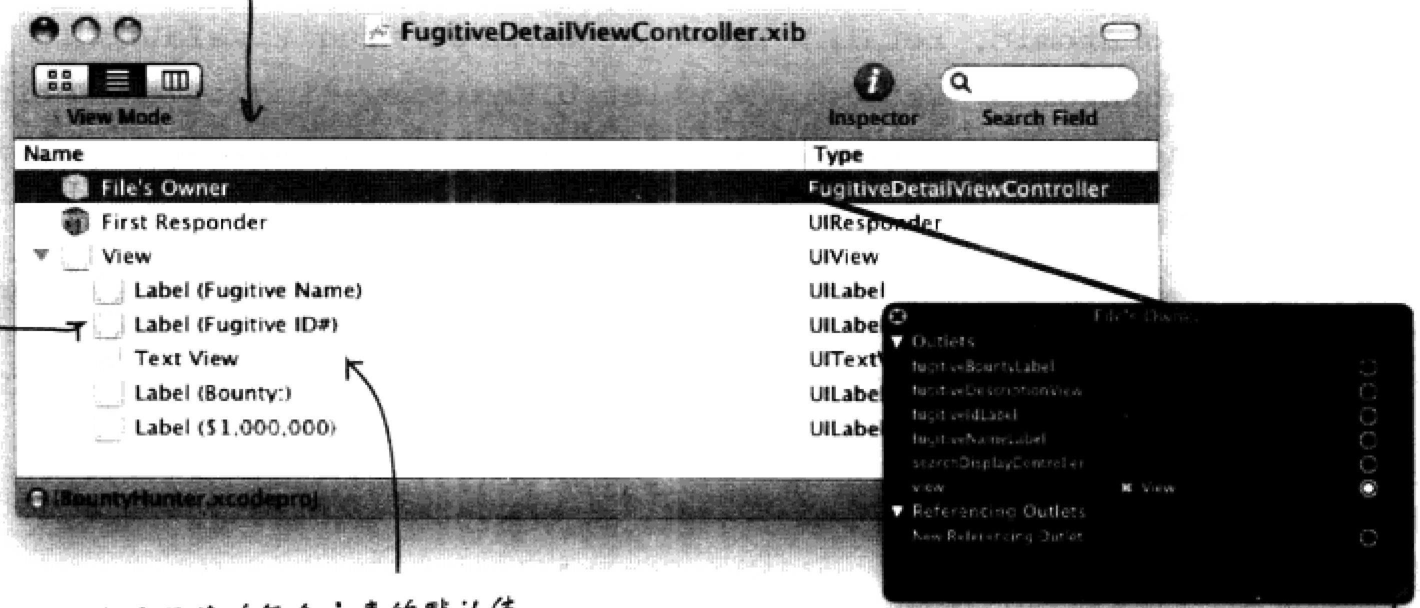
FugitiveDetailViewController.m

【拼吾爱-最新编程资源的分享下载站】

长篇练习答案

现在在Interface Builder中构造视图。

这里就是细节视图中所有组件的最终列表。

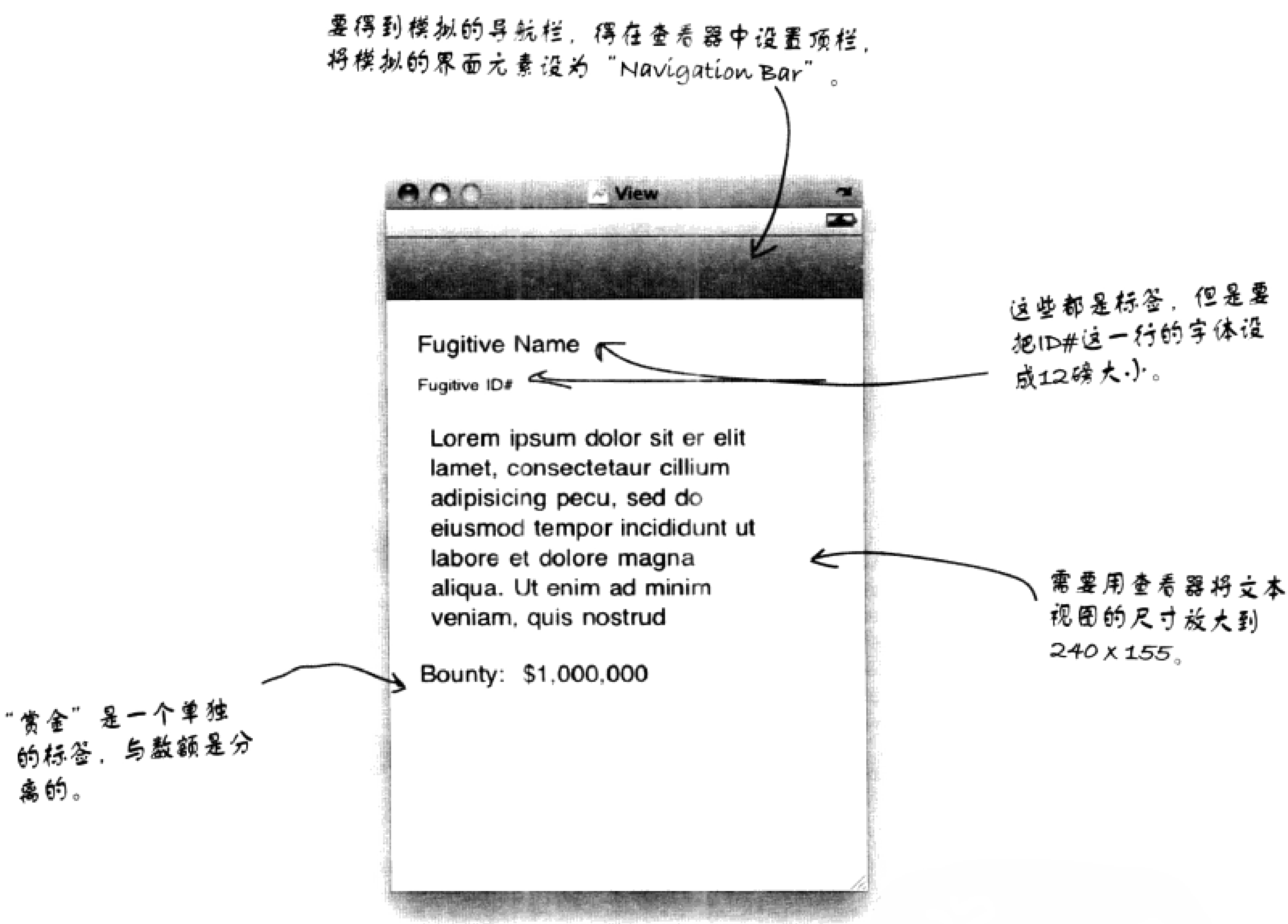


确保添加的元素都是主视图的子节点。

用查看器修改每个元素的默认值，分别改为“逃犯姓名”、“逃犯ID”等。

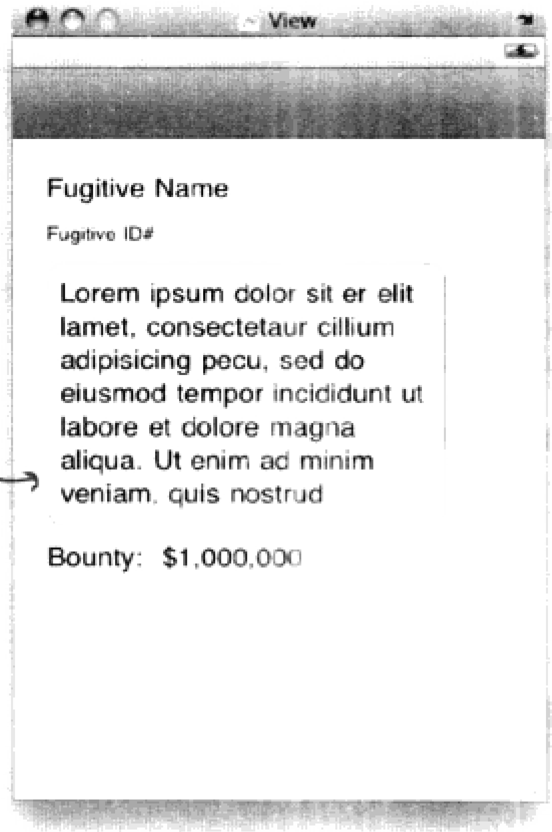
确保你将这些都连接好了…

【拼吾爱-最新编程资源的分享下载站】



极客点滴

我们打算给这个视图加一些美化修饰。现在这样也不错，但是还可以加一些iPhone特有的酷感。



在UITextView的正上方，加一个圆角矩形按钮。

我们用这个按钮来作为一个边框。要做出这个效果，需要以下两个步骤：

1. 点击圆角矩形按钮，然后在菜单中依次选择Layout → Send to Back选项。

2. 保持按钮的选中状态，用查看器取消选中enabled box点选框（在Content的下面）。

【拼吾爱-最新编程资源的分享下载站】

练习

现在，只要从逃犯列表中向细节视图填充数据就行了。根据以前我们在DrinkMixer中的做法，你应该知道怎么做了。

1 其他文件需要知道细节视图的存在。

在一些实现文件中，你需要`#import` FugitiveDetailViewController.h。

你应该能想到是哪个文件。

2 细节视图需要被调用。

在同一个实现文件中，表格视图需要一些代码，当选中某行的时候会用到。这些代码应该与我们在DrinkMixer中用到的类似。

3 需要向字段中填充数据。

细节视图的代码需要向已有的字段中填充数据，这些数据来自Fugitive.h和Fugitive.m类中的成员变量以及Core Data代码。填充过程要放在`viewWillAppear`中：

这些只是例子，但应该可以明白地表示出该怎么做了。

```
fugitiveNameLabel.text = fugitive.name;
fugitiveIdLabel.text = [fugitive.fugitiveID stringValue];
```

4 将它们连接到一起。

回到IB中，将你的表格视图与其代理联系起来。

【拼吾爱-最新编程资源的分享下载站】

练习答案

现在，只要填充细节视图就行了。从以前的经验中，你应该已经学会如何做了。

1. **其他文件需要知道细节视图的存在。**
在一些实现文件中，你需要#import FugitiveDetailViewController.h。

把这个加到FugitiveListViewController.m文件的顶部。

2. **细节视图需要被调用。**
在同一个实现文件中，表格视图需要一些代码，当选中某行的时候会用到。这些代码应该与我们在DrinkMixer中用到的类似。

```
-   (void)
tableView:(UITableView*)tableView didSelectRowAtIndexPath: (NSIndexPath
*)indexPath {


      FugitiveDetailViewController *fugitiveDetailViewController
= [[FugitiveDetailViewController alloc] initWithNibName:
@"FugitiveDetailViewController" bundle:nil];
      fugitiveDetailViewController.fugitive = [self.items
objectAtIndex:indexPath.row];
      [self.navigationController pushViewController:
fugitiveDetailViewController animated:YES];
      [fugitiveDetailViewController release];
  }
```

这里我们告诉细节视图控制器该显示哪个逃犯。

FugitiveListViewController.m

【拼吾爱-最新编程资源的分享下载站】

❸ 需要向字段中填充数据。

```
#import "FugitiveDetailViewController.h"
#import "Fugitive.h"
@implementation FugitiveDetailViewController
@synthesize fugitive, fugitiveNameLabel, fugitiveIdLabel,
fugitiveDescriptionView, fugitiveBountyLabel;

-(void) viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];

    fugitiveNameLabel.text = fugitive.name;
    fugitiveIdLabel.text = [fugitive.fugitiveID stringValue];
    fugitiveDescriptionView.text = fugitive.desc;
    fugitiveBountyLabel.text = [fugitive.bounty stringValue];
}
```

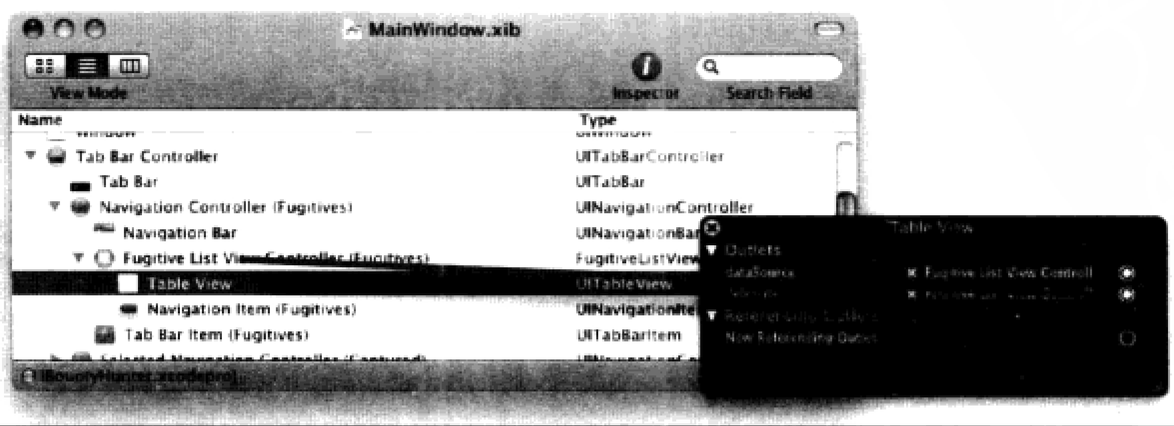
我们打算在Fugitive类中访问字段。我们需要告诉编译器这一点。

在这两个声明的末尾加上stringValue，因为它们是NSNumber和NSDecimalNumbers而不是字符串。

FugitiveDetailViewController.m

❹ 将它们连接到一起。

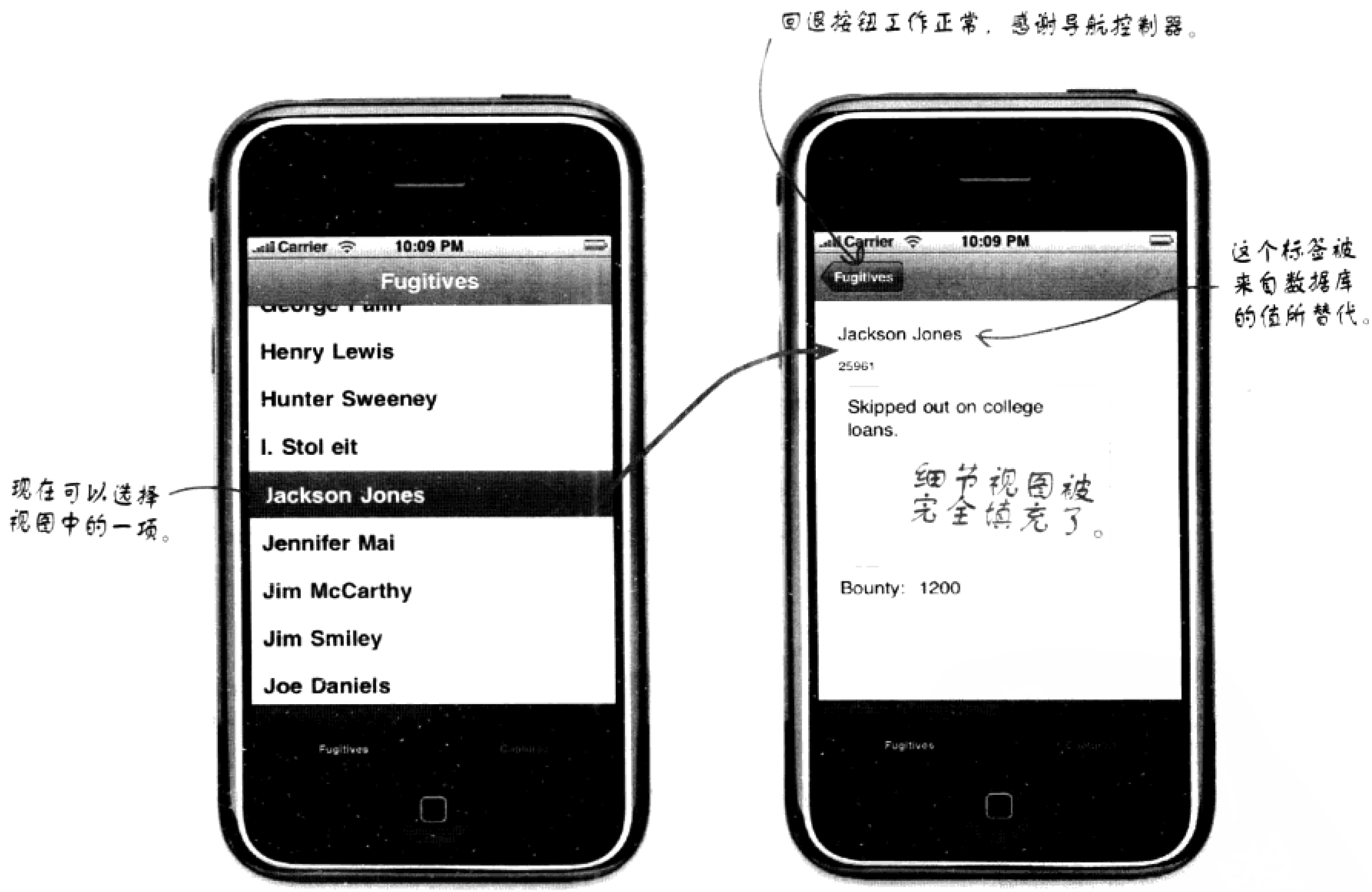
在IB中，逃犯列表视图控制器下表格视图的代理需要与该视图控制器连接在一起。



【拼吾爱-最新编程资源的分享下载站】

运行测试

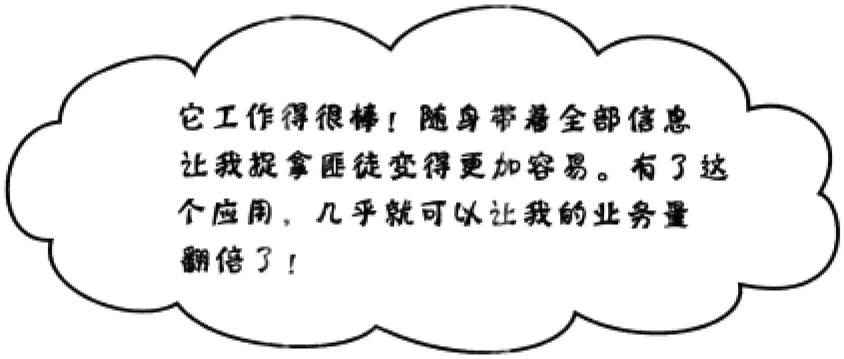
填充好细节视图之后，你可以看到每个逃犯的信息。



全部工作正常！

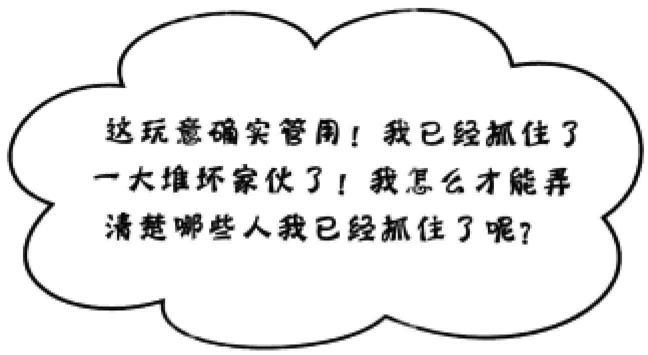
【拼吾爱-最新编程资源的分享下载站】





太棒了！

过了几个星期，鲍勃又回来了，带着新的请求……



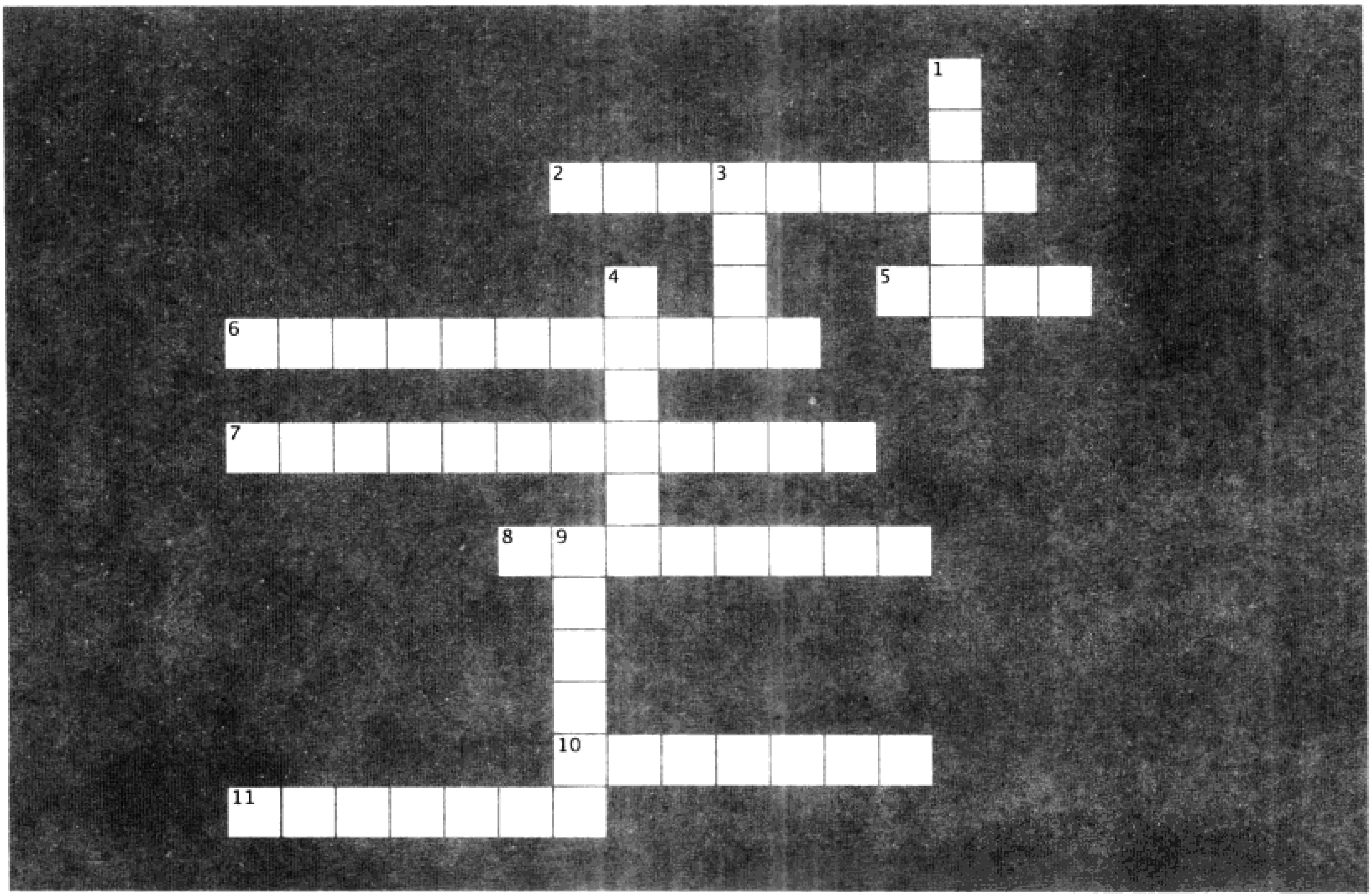


未完待续……

【拼吾爱-最新编程资源的分享下载站】

Core Data纵横字谜

Core Data中有许多术语；让我们看看你记住了没有！



横向提示

2. 每个应用都有一个________目录。
5. NS________Descriptor可以说明数据该如何排序。
6. 在Core Data堆栈的中间，是持久存储______。
7. ________模板相当简单。
8. ________可以管理不同类型的数据。
10. 受控对象________位于Core Data堆栈的顶部。
11. NSFetch________描述了一次搜索。

纵向提示

1. 持久对象存储位于Core Data堆栈的________。
3. Core Data有________和重做功能。
4. ________控制器适合用于切换视图。
9. 受控________模型对实体进行描述。

【拼吾爱-最新编程资源的分享下载站】



你的Core Data工具箱

你已经消化吸收了第7章，现在你已经把Core Data加入到了你的工具箱中。 如果你想得到一份完整的书中功能提示的列表，请访问http://www.headfirstlabs.com/iphonedev。

数据模型

用于实体，实体也同样具有属性。

可以直接在Xcode中编辑。

有几种不同的数据类型。

标签栏

每个标签都代表了一个单独的视图。

标签对于非结构层次化的任务很合适。

Core Data

提供了一个堆栈，可以替你管理数据。

可以管理不同类型的数据。

非常适合用来进行内存管理和变化跟踪。

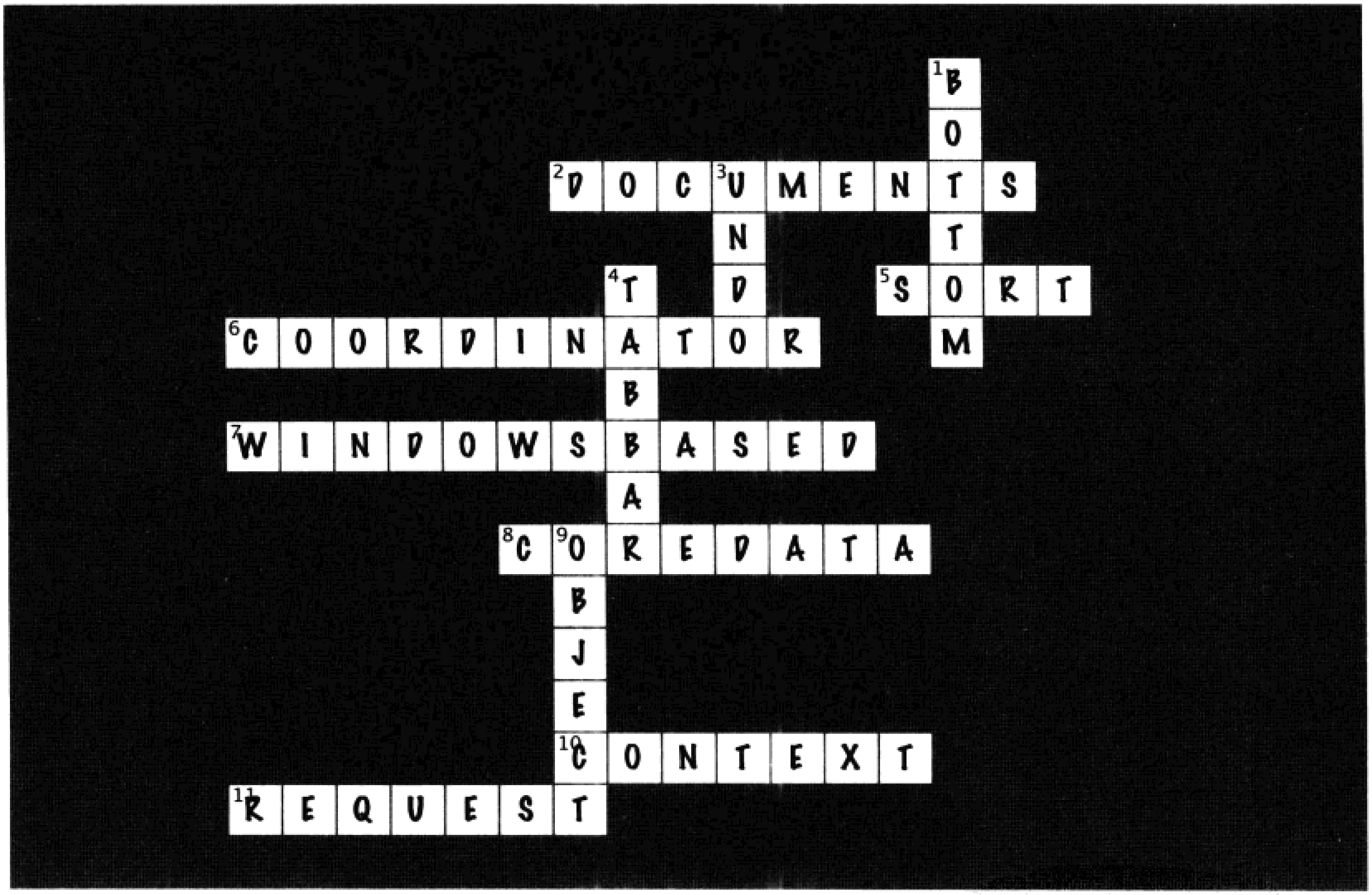
关键要点

- Core Data是一个持久化框架，提供了加载、保存、版本控制和撤销－重做功能。
- Core Data可以建立在SQLite数据库之上，也可以是二进制文件或者临时内存。
- 受控对象模型定义了实体，Core Data将会使用这些实体进行工作。
- 受控对象上下文是我们访问数据的入口点。它会跟踪活动的受控对象的状态。
- 受控对象上下文是Core Data堆栈的一部分，负责处理数据的读写。

【拼吾爱-最新编程资源的分享下载站】

Core Data纵横字谜答案

那么，你记住了所有这些词语了吗？



横向提示

2. 每个应用都有一个__________目录。[DOCUMENTS]

5. NS_________Descriptor可以说明数据该如何排序。[SORT]

6. Core Data堆栈的中间，是持久存储_______。[COORDINATOR]

7. __________模板相当简单。[WINDOWSBASED]

8. __________可以管理不同类型的数据。[COREDATA]

10. 受控对象__________位于Core Data堆栈的顶部。[CONTEXT]

11. NSFetch__________描述了一次搜索。[REQUEST]

纵向提示

1. 持久对象存储位于Core Data堆栈的__________。[BOTTOM]

3. Core Data有_________和重做功能。[UNDO]

4. _________控制器适合用于切换视图。[TABBAR]

9. 受控__________模型对实体进行描述。[OBJECT]

【拼吾爱-最新编程资源的分享下载站】

8 用Core Data迁移并优化数据

一切都改变



我们的应用工作得非常好。iBountyHunter成功地加载了鲍勃需要的数据，让他可以用一种方便的方式来查看逃犯。但是如果必须改变数据的话该怎么办？鲍勃想要一些新功能，这会对数据模型造成什么样的影响呢？在这一章中，你将学到如何处理对数据模型进行改变，以及如何利用更多的Core Data功能。

【拼吾爱-最新编程资源的分享下载站】

鲍勃需要一些文书工作



鲍勃需要记录更多的信息。

鲍勃必须记录自己的工作状态，这样他才能获得报酬。这意味着我们需要在某处存储逃犯被捕的日期和时间，然后用这些信息构造一个被捕视图……

还记得我们在上一章里构建的那个被捕视图吗？

我们应该怎样改进iBountyHunter才能处理这些新信息呢？

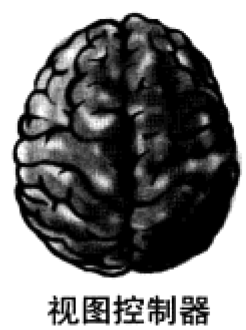
【拼吾爱-最新编程资源的分享下载站】

拿起笔来

我们需要找出应该如何改进iBountyHunter，来处理新的数据。查看我们的应用程序，写出各个部分是否需要修改，该如何修改。









我们该从哪里着手？

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

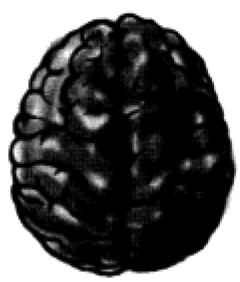
我们需要找出应该如何改进iBountyHunter，来处理新的数据。查看我们的应用程序，写出各个部分是否需要修改，该如何修改。



- 在某处将逃犯标记为已被捕。
- 显示被捕的日期和时间。
- 填充被捕列表。



- 给逃犯加一个被捕的标记。
- 为逃犯添加被捕时间。
- 为逃犯添加被捕日期。



视图控制器

- 填写被捕数据的日期和时间。
- 被捕视图中只显示被捕的逃犯。



- 为数据添加有关修改的信息，以便在应用中显示。

我们该从哪里着手？由于有关新数据的几乎一切内容都需要我们来添加，就先处理对象模型吧；然后再修改其余部分。

一切都源自我们的数据模型

按照我们在练习中的考虑结果，实体Fugitive需要添加几个字段：日期和时间，还有一个用来表明逃犯是否已被抓获的标记。数据库是根据数据模型构建的，所以我们只要修改数据模型就可以加入我们需要的信息。Core Data的日期类型同时包含了日期和时间，所以我们只需要在Fugitive实体中再加两个属性就可以了：



拿起笔来

更新数据模型，加入新的被捕相关字段。

- [ ] 用数据模型编辑器更新模型，加入两个新字段。

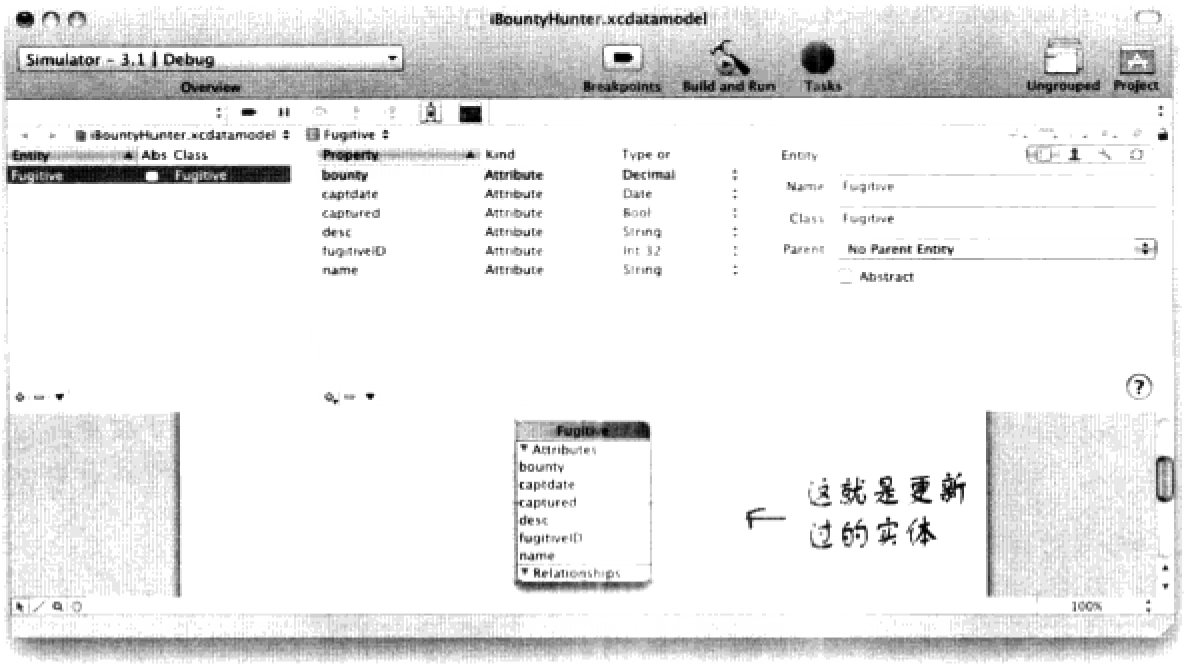
- [ ] 更新模型之后，你需要删除两个旧的fugitive类文件，并生成包含新字段的新文件。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

用Xcode自带的工具快速完成这些修改。

- [x] 用数据模型编辑器更新模型，加入两个新字段。



这就是更新过的实体

captured
- Boolean
- 必需
- 默认为NO

确保你的属性与这里的一致。

captdate
- Date
- 可选

- [x] 更新模型之后，你需要删除两个旧的fugitive类文件，并生成包含新字段的新文件。

```
#import <CoreData/CoreData.h>

@interface Fugitive :  NSManagedObject
{
}

@property (nonatomic, retain) NSDecimalNumber * bounty;
@property (nonatomic, retain) NSNumber * captured;
@property (nonatomic, retain) NSNumber * fugitiveID;
@property (nonatomic, retain) NSDate * captdate;
@property (nonatomic, retain) NSString * name;
@property (nonatomic, retain) NSString * desc;

@end
```

这两个字段是新生成的。

Fugitive.h

【拼吾爱-最新编程资源的分享下载站】


```
#import "Fugitive.h"

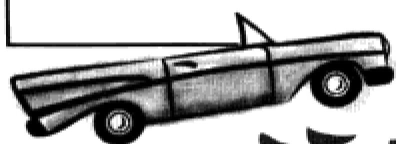
@implementation Fugitive

@dynamic bounty;
@dynamic captured;
@dynamic fugitiveID;
@dynamic captdate;
@dynamic name;
@dynamic desc;

@end
```

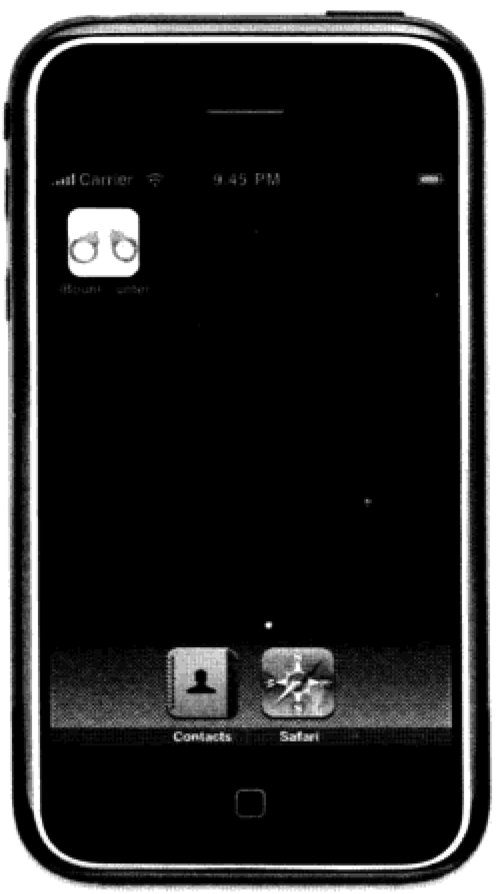
Fugitive.m

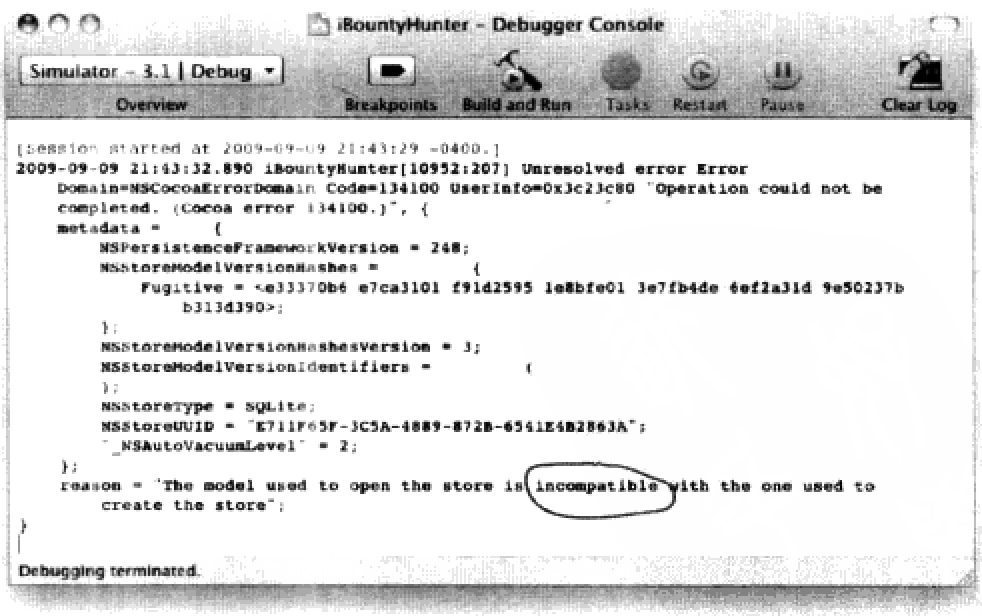
新字段是作为动态属性添加的，就像以前那些一样。



运行测试

当你完成修改之后，继续运行iBountyHunter。





应用崩溃了！

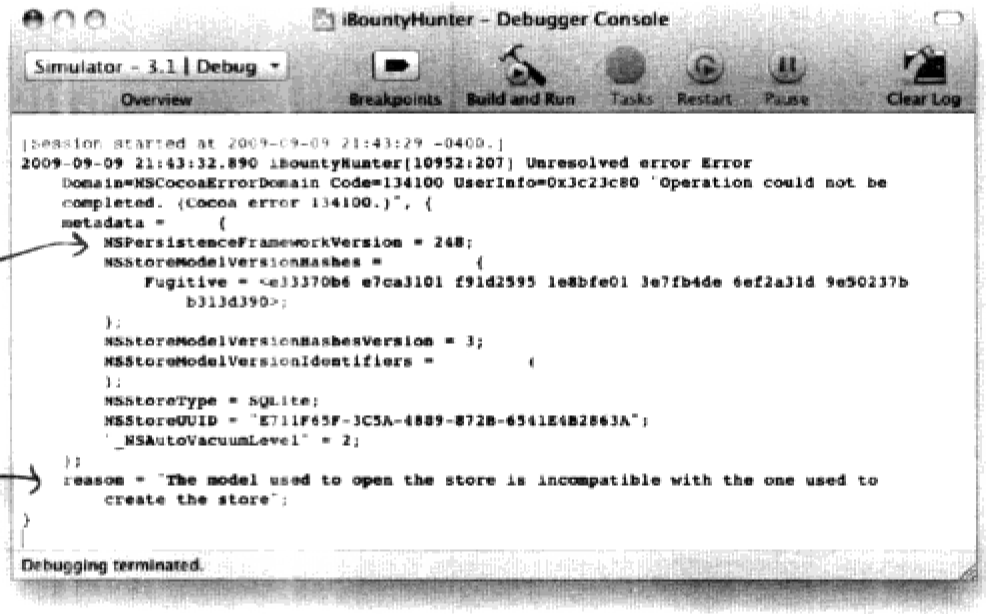
脑力练习

为什么应用会崩溃呢？

【拼吾爱-最新编程资源的分享下载站】

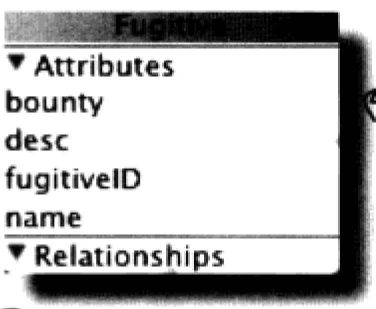
数据还没有更新

如果你仔细查看终端中关于崩溃的报告，就可以发现哪里出错了……

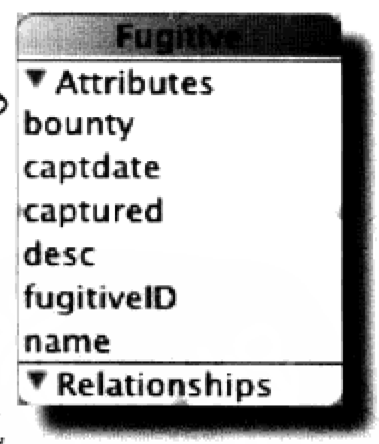


这个错误看来是来自Core Data。

错误信息说我们的数据模型与用来创建数据的模型不兼容。



这个数据模型与我们用来实际写入数据的模型不同。



我们原来的Fugitive实体只有4个属性…

…但当前的Fugitive实体多了两个：被捕日期和被捕标记。

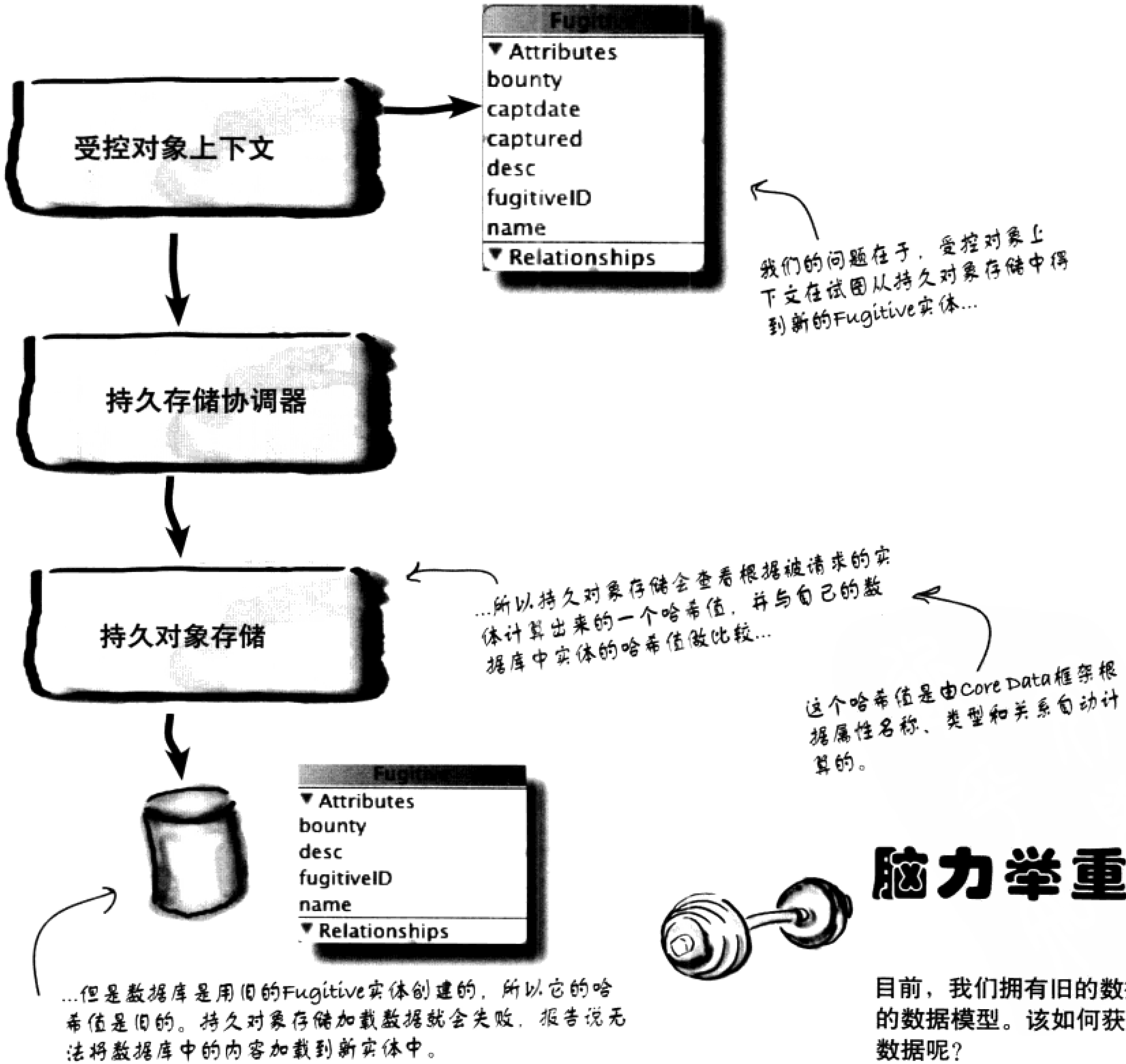
Core Data发现我们的数据库与模型之间不匹配

我们向Fugitive实体添加新字段引起了这个问题。我们起初的逃犯数据库是用旧模型创建的，而Core Data根本不知道到哪里去找到这些新字段。为了避免对数据造成破坏，它并不会冒险访问数据，而是报告错误并退出应用程序。这样很好，但是我们仍然需要想办法修正这个问题。

【拼吾爱-最新编程资源的分享下载站】

数据迁移是一个常见问题

发现自己需要添加新数据，或者要改变旧的数据存储方式，在应用程序开发中是一种相当常见的问题。但是，虽然它很常见，却并不意味着很简单。Core Data努力确保不会破坏也不会丢失任何数据，所以我们必须告诉它该怎么处理我们的新Fugitive实体。



脑力举重

目前，我们拥有旧的数据和新的数据模型。该如何获得新的数据呢？

【拼吾爱-最新编程资源的分享下载站】

我们需要将旧数据迁移到新模型中

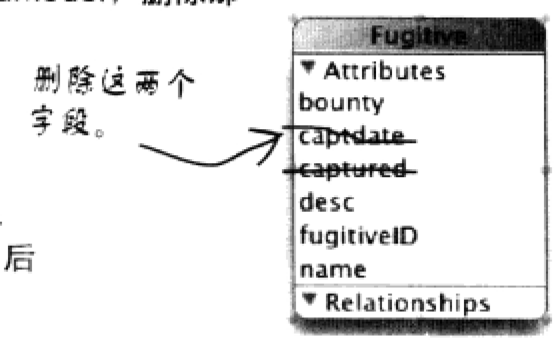
我们对数据模型做出了修改，但是还需要让Core Data堆栈中上上下下的所有层次都能够处理这些修改。要做到这一点，我们需要进行数据迁移。

要迁移一件东西，你需要弄清楚此物的来龙去脉。Core Data需要新旧两个数据模型，才能令数据迁移对整个堆栈生效。我们需要用一种新的方式来修改数据模型，而不是仅仅改变旧的模型。让我们撤销之前所做的改动，以便能够再次从数据库中加载数据。



拆除数据模型

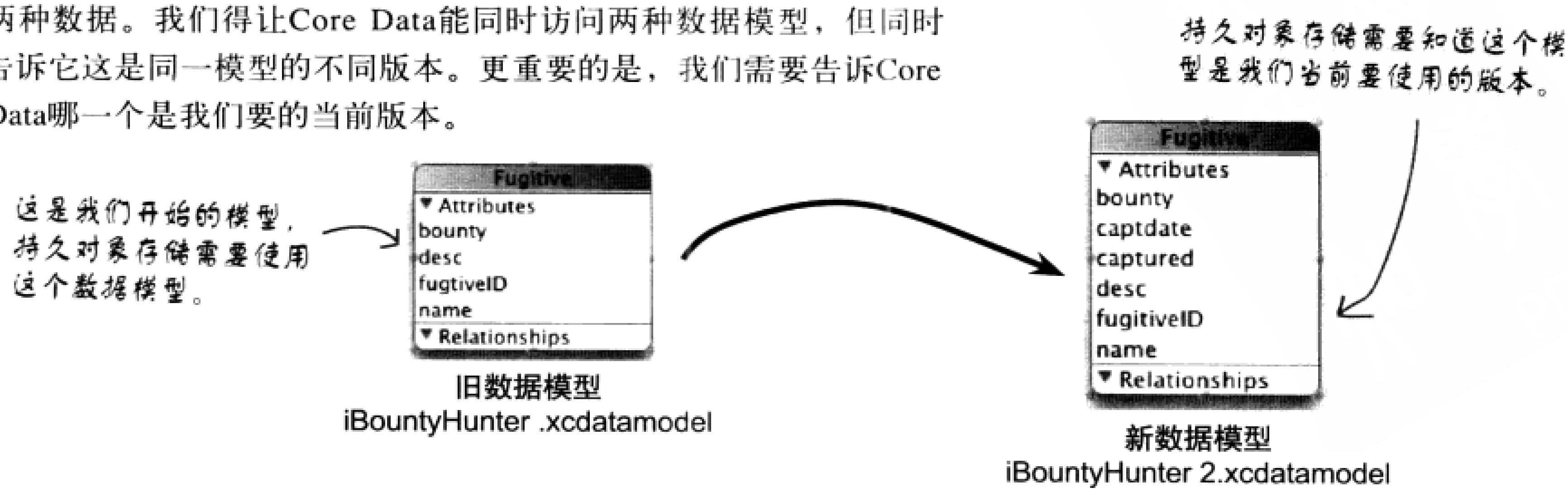
为了让数据模型能具有一个起点和终点，我们需要回退到原来的iBountyHunter.xcdatamodel，删除那两个新字段——仅仅是目前而已。



然后回去检查应用是否再次工作正常！这样会节省许多时间，避免以后的麻烦。

我们的两个模型需要使用不同的版本

手工修改数据模型十分容易，但是Core Data需要能够同时使用新旧两种数据。我们得让Core Data能同时访问两种数据模型，但同时告诉它这是同一模型的不同版本。更重要的是，我们需要告诉Core Data哪一个是我们要的当前版本。



【拼吾爱-最新编程资源的分享下载站】

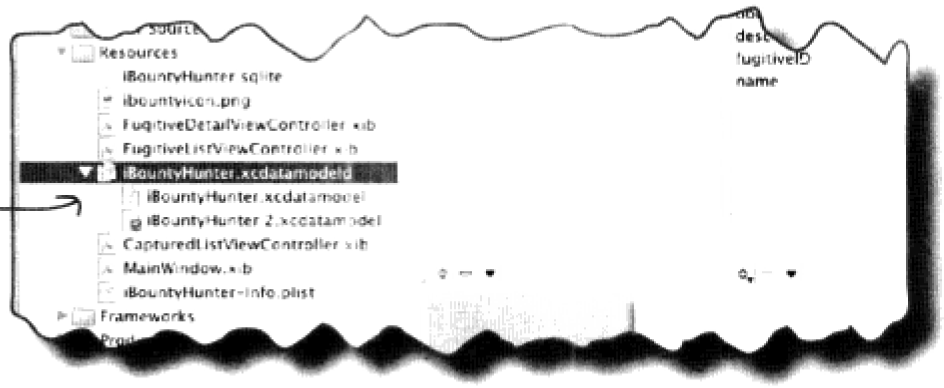
Xcode令数据模型版本控制变得轻松

幸运的是，用Xcode创建一个数据模型的新版本是相当容易的：

1. **选中iBountyHunter.xcdatamodel。**
   然后在菜单中选择**Design → Data Model → Add Model Version**菜单项。这样就会生成一个名为iBountyHunter.xcdatamodeld的新目录。在这个目录下，会有数据模型的两份拷贝。

2. **设置当前版本。**
   在iBountyHunter.xcmodeld目录内，选择iBountyHunter 2.xcdatamodel，这将会作为我们的新版本。在菜单项中依次选择**Design → Data Model → Set Current Version**。

这就是文件列表最终的样子。



3. **修改新数据模型。**
   选择iBountyHunter 2.xcdatamodel并编辑该数据模型，重新像以前一样加回captdate和captured这两个字段。现在老版本被保留下来，而修改也各就各位了。

极客点滴

正常情况下，你还会需要删除并重新生成Fugitive类，不过由于我们对新文件做的修改是一样的，所以生成的类也会相同。

应用怎样在两个版本之间做映射呢？

【拼吾爱-最新编程资源的分享下载站】



吉姆：嗯。我猜我们需要写一大堆迁移代码之类的东西。

乔：为什么？

吉姆：我觉得我们得告诉Core Data如何从旧版本的数据中取得新数据，对吗？

弗兰克：哎呀，实际上，我想咱们可以让它自动完成。

吉姆：什么？

弗兰克：Core Data有一个功能，可以让应用得知两个模型的存在，这样就可以替你迁移数据。

吉姆：太好了！数据是什么时候被实际迁移的？

弗兰克：在运行时，当持久对象存储看到数据是旧的格式时。这就意味着我们仅仅需要一些告诉iBountyHunter实际执行迁移的代码。

乔：好吧，这么说代码中有一部分会是自动生成的，一部分需要我们来添加。

吉姆：这真是好极了；这样就可以只修改我们需要的部分了吧？

弗兰克：有一些数据的变化是Core Data可以自动处理的，比如添加新属性。更复杂的数据变化就需要手工处理了。

乔：是的，这里写着，如果我们增加属性或者改变属性的可选状态，就可以自动迁移。

吉姆：剩下的怎么办？

弗兰克：剩下的就要麻烦一些了——有时候可以、有时候不行。

乔：那么，怎么才能迁移我们的数据呢？

【拼吾爱-最新编程资源的分享下载站】

Core Data可以“轻量”迁移数据

轻量级数据迁移是Core Data的一种强力工具，让你不需要映射模型就可以干净利落地更新底层数据，以匹配一个新的数据模型。它的作用范围仅限于基本的数据更改：增加新属性，将一个必需属性变为可选的，或者将可选属性变为有默认值的必需属性。它也可以有限地处理属性更名，但这部分更具技巧性。

自动数据迁移发生在运行时，这意味着你的应用需要了解迁移将会发生，这样数据才能被迁移。你应该在应用代理中进行这项工作：

```
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {
      if (persistentStoreCoordinator != nil) {
       return persistentStoreCoordinator;
   }

   NSURL *storeUrl = [NSURL fileURLWithPath: [[self applicationDocumentsDirectory]
stringByAppendingPathComponent: @"iBountyHunter.sqlite"]];
      NSError *error = nil;
   persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedOb
jectModel:[self managedObjectModel]];
      NSDictionary *options = [NSDictionary dictionaryWithObjectsAndKeys:
  [NSNumber numberWithBool:YES], NSMigratePersistentStoresAutomaticallyOption,
  [NSNumber numberWithBool:YES], NSInferMappingModelAutomaticallyOption, nil];
      if (![persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType
configuration:nil URL:storeUrl options:options error:&error])
```

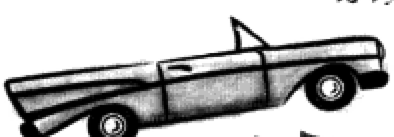
记得吗？默认情况下Core Data将会加载应用束中的所有对象模型。也就是说，它会同时见到旧版本和当前版本的模型。



iBountyHunterAppDelegate.m

我们用这个代替了nil：将options传给持久存储协调器。

要启用轻量级迁移，我们需要做的只不过是把这个打开。

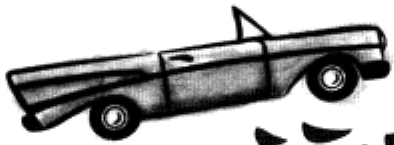


运行测试

将上面的代码加入应用代理后，构建并调试...

如果这儿出了问题，先试一下Build->Clean，再构建并调试。很奇怪，首次建立新版本模型时，Xcode并不总是能正确重编译，不过先清理一遍就好了。

【拼吾爱-最新编程资源的分享下载站】



运行测试



干得漂亮！现在用新数据模型工作也完全正常了。

【拼吾爱-最新编程资源的分享下载站】



持久对象存储全接触

本周访谈：
你真的还有什么能力吗？

Head First: 你好，持久对象存储，如果我简单点管你叫POS你介意吗？

持久对象存储： 我觉得你最好别这样叫我。就叫我“存储”好了。

Head First: 好的，存储，按照我的理解你是Core Data堆栈的一部分？

存储： 是的——实际上是最重要的部分。我的任务就是实际读写数据。

Head First: 对啊，你就是负责在一堆不同的格式间进行翻译的人。

存储： 正是如此。只要你使用Core Data，就不需要真正了解数据是存在一个简单的文件中，还是在复杂的数据库里。你只要让我读写一组数据，我就能搞定它。

Head First: 这很方便。不过我知道你挺特别。我听说你对变化适应不良。

存储： 我觉得你没搞清整个背景。知道吗？我的工作就是保证数据被完全正确地保存和加载。

Head First: 我知道这个，不过还是得说，小修改没问题，对吧？

存储： 确实——只要你确认真要这么做就行。你得告诉我看到的数据是什么样，然后再说明要得到什么样的返回值。知道了这些就可以推算数据之间的差异并进行映射，我就会处理其余的事务。

Head First: 那么你实际迁移了数据，还是只在加载时进行转换？

存储： 哦，我实际迁移了数据。这就是最酷的地方。像二进制文件这样的简单存储，就会用迁移后的数据建立一个新文件。但我用的是SQLite数据库，一般我都可以进行就地迁移。并不需要加载数据，而整个迁移也几乎是即刻完成的。

Head First: 真棒！我觉得轻量级迁移就是一种“傻瓜”式迁移。

存储： 哦，不能完全这么说，如果你可以让我通过轻量级迁移完成迁移，那可以说就是这种方式。如果你需要进行一些更复杂的工作，比如将一个旧属性分割成两个新属性，或者改变某个东西的类型等，你就得帮我一把了。

Head First: 那大家是得通过代码来帮助你吗？

存储： 差不多吧。基本上来说，你需要再多给我一个模型，一个映射模型。这个模型会告诉我该如何将数据从旧格式变成新格式。

Head First: 嗯，好吧，有道理。我猜这也可以处理变量更名吧？

存储： 实际上，大多数情况下我自己也可以处理这个变化，只要你告诉我旧名字是什么就行。如果你查看数据模型中一个属性的细节，就可以给出属性的旧名字。如果有旧名字，而我又要进行迁移，我就也可以处理更名。

Head First: 喔，你可不像我想象的那么无趣啊……

存储： 我猜我还得谢谢你哪。

【拼吾爱-最新编程资源的分享下载站】

有问必答

问： 数据模型能有多少个版本？

答： 你需要多少就可以有多少。只要你开始增加版本，就需要跟踪当前版本的状态，这样受控对象模型才能知道你在请求一个实体时到底想要什么。只要保留所有的旧版本，Core Data就可以从任意以前的版本迁移到当前版本。

问： 为某个东西改名的话，什么时候适用轻量级迁移？什么时候不适合？

答： 你可以为变量改名，只要不改变类型就行。要为属性改名，可以在Xcode中点击属性面板上的小扳手图标，并将“renaming identifier”指定为旧的属性。Core Data会自动根据这里指定的名称处理数据的迁移。

问： 我能否利用数据迁移将其他一些格式的数据导入到Core Data？

答： 不行。迁移（无论是否轻量级）只能用于已有的Core Data数据。如果你有想要导入Core Data的历史遗留数据，你需要自己完成。通常，你需要做的只是用自己的代码读入遗留数据，创建一个新的NSManagedObject来保存这些数据，将数据填充到新对象中，再用Core Data保存对象就可以了。不是很优雅，不过管用。还有几种方法，如果你要迁移大量数据或者流数据（比如来自网络的数据）的时候可以考虑使用。看看苹果公司的一篇名为《利用Core Data高效导入数据》（Efficiently Importing Data with Core Data）的文档，里面有更多细节信息。

问： 用轻量级迁移和自己手工迁移数据有区别吗？

答： 尽量使用轻量级迁移。虽然它不能适用于所有情况，但如果能用，而你又使用了SQLite存储，Core Data可以优化迁移过程。当使用轻量级迁移时，迁移花费的时间可以非常非常短。

问： 要是我不能用轻量级迁移怎么办？

答： 那就需要创建一个映射模型。你可以在Xcode中创建此模型，做法是在菜单中选择Design→Mapping Model，然后挑选要做映射的两个模型。你还需要选择源实体和属性，然后选择目标实体和属性。如果有必要，你可以输入自定义的表达式来做数据转换。要了解更多关于映射模型的信息，请查看苹果公司关于Core Data数据迁移的文档。

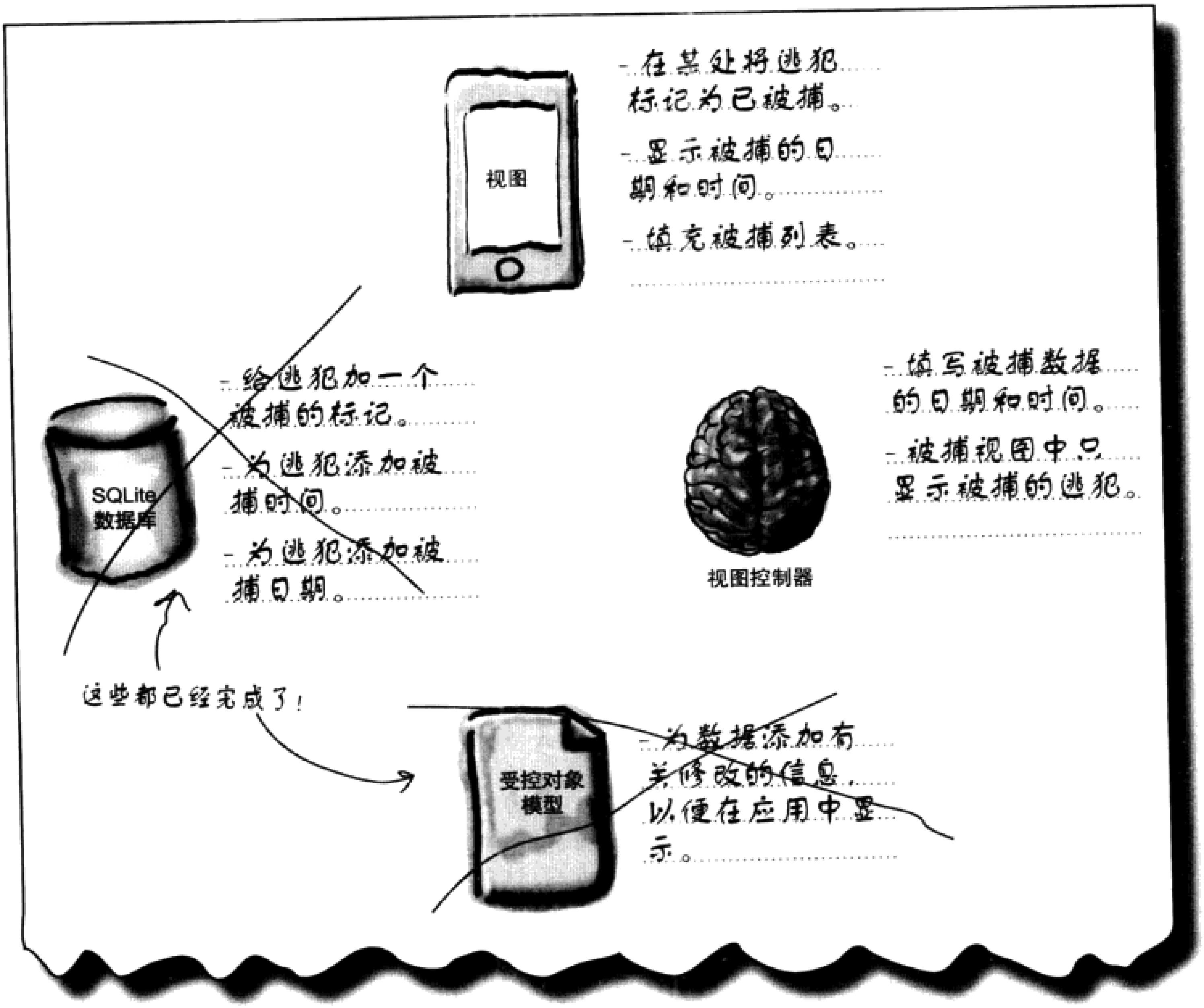
问： Xcode让我在一个属性的版本设置中输入一个哈希修正量。这是干什么用的？

答： Core Data会用属性信息为实体计算出一个哈希值，以便确定自从数据存储创建之后模型是否有改变。然而，也有可能你需要改变数据存储的方式，却并未实际改变数据模型。举个例子，假如说你总是以秒为单位来存储时间值，但是后来又决定要用毫秒来存储。你可以继续用一个整数来存储这个值，但是用一个哈希修正量让Core Data明白你希望这两个模型被当成不同版本来对待，并在运行时使用你的迁移代码。

关键要点

- 轻量级自动迁移同时需要数据模型的两个版本才能工作。
- 自动迁移可以不加载数据就更改一个SQLite数据库。
- 数据的迁移发生在运行时。
- 你可以用轻量级迁移来增加变量，将必需变量变为可选，将可选变量变为有默认值的必需项，还能进行一些更名工作。

【拼吾爱-最新编程资源的分享下载站】



脑力练习

我们需要对UI做什么样的改动，才能加入抓捕信息？

【拼吾爱-最新编程资源的分享下载站】

鲍勃有一些设计方面的建议



但是鲍勃的草图有一些问题……

【拼吾爱-最新编程资源的分享下载站】

拿起笔来

鲍勃的视图还需要一些改进。作为一个经验丰富的iPhone开发者，你应该可以提出一些更好的UI设计方案。现在你该帮他一把了。

1. 鲍勃的视图如果原样不变的话，能实际用在应用里吗？（圈起答案） **是** **否**

2. 如果不能，为什么？…………………………………………

…………………………………………

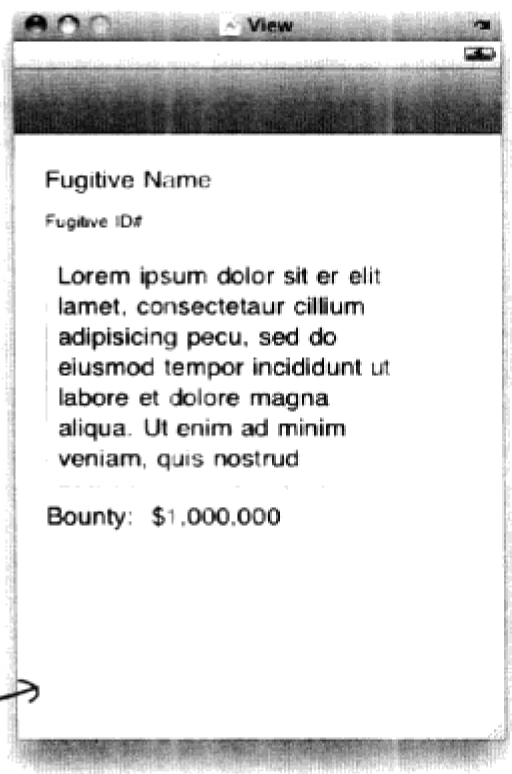
…………………………………………

3. 要正确地实现这个视图，你需要了解哪些数据是可以修改的。哪些数据可以让用户编辑，处理输入用什么方式最好？…………………………………………

…………………………………………

…………………………………………

4. 用草图画出你计划的最终细节视图：



别忘了下面这儿还有标签栏控制器呢。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

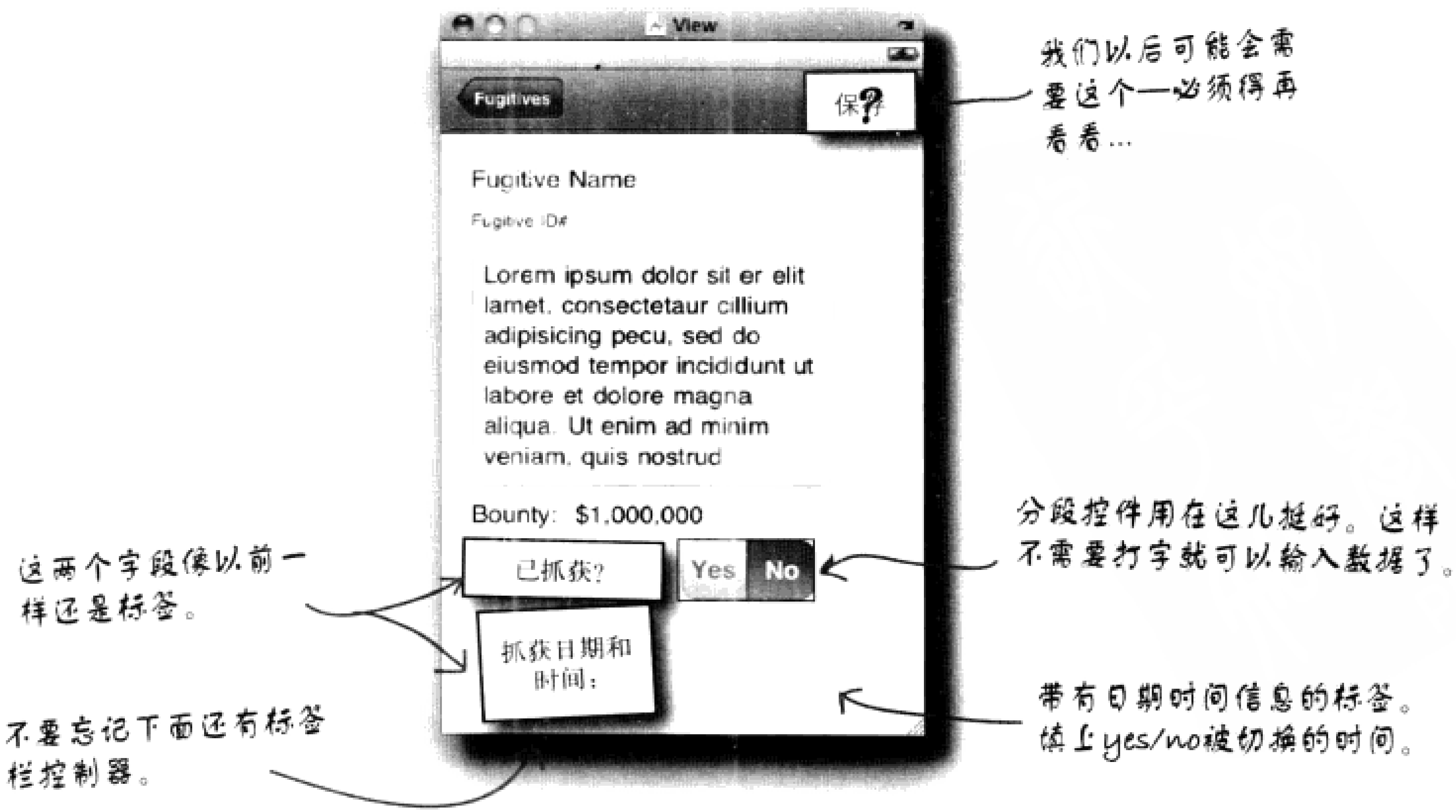
现在你经过详细考虑，得出了设计结论，那么细节视图看起来应该是什么样子？

1 鲍勃的视图如果原样不变的话，能实际用在应用里吗？（圈起答案） 是 否（已圈出）

2 如果不能，为什么？在鲍勃想放取消按钮的地方，我们已经有一个回退按钮了。让用户输入“是”或“否”并输入日期和时间也不是好的UI。

3 要正确地实现这个视图，你需要了解哪些数据是可以修改的。哪些数据可以让用户编辑，处理输入用什么方式最好？唯一可以改变的数据是被捕标志字段和被捕日期和时间。由于被捕标志是一个布尔值，所以用开关或者其他类型的控件会比输入效果更好。由于鲍勃会在抓获坏人的时候点击控件，所以我们可以从iPhone获取当前日期和时间，省得输入。

4 用草图画出你计划的最终细节视图：



【拼吾爱-最新编程资源的分享下载站】

练习

按照你的需要向细节视图中添加新增的这些控件。

❶ **在Interface Builder中打开FugitiveDetailViewController.xib。**

然后添加你需要的视觉元素：3个标签以及分段控件。你还需要加上一个模拟的标签栏来保证所有东西的位置都合适。现在先不要担心保存按钮的问题。

❷ **在FugitiveDetailViewController.m (和.h)中，加入属性与初始化代码。**

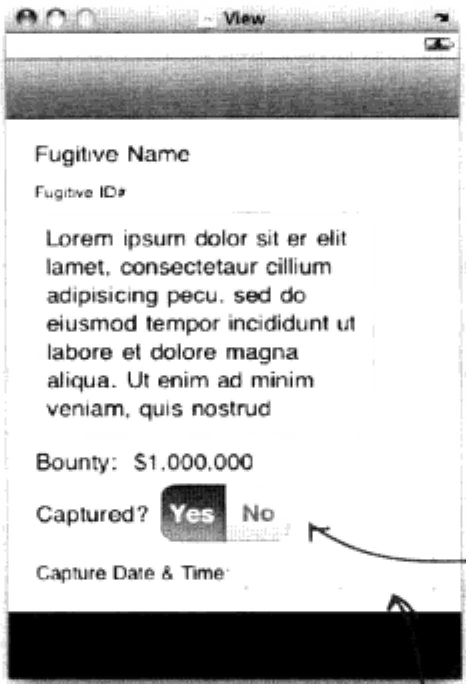
现在所有这些界面元素都有了，该在Xcode里给它们制造后端了，不过现在还不需要操心如何去连接它们……

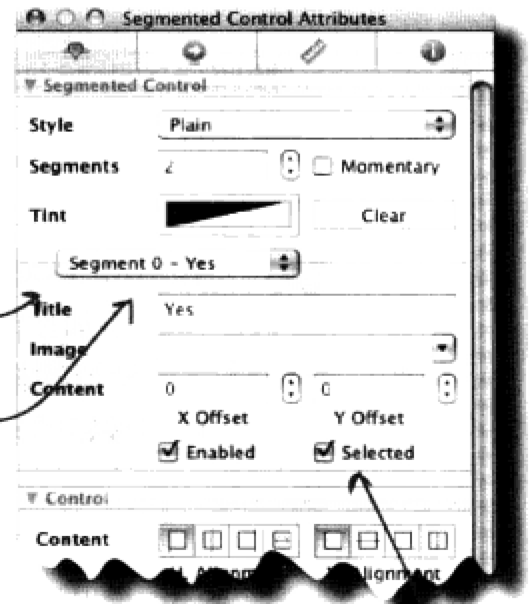
【拼吾爱-最新编程资源的分享下载站】

练习答案

这里是为视图添加的内容，以及对其进行支持的代码。

1. 在Interface Builder中打开FugitiveDetailViewController.xib。





切换这个选项可以访问控件的另一半。

需要对分段控件进行配置（默认标题会是first / second）。

这个标签将显示被捕日期和时间，但只有开关被切换到yes才会有内容。

要保证"Segment 0 - Yes"被选中，同时两个分段都是开启状态。我们马上就要用到它。

2. 在FugitiveDetailViewController.m (和.h)中，加入属性与初始化代码。

```
UISegmentedControl *capturedToggle;
UILabel *capturedDateLabel;
```

在@interface之内加上这些。

```
@property (nonatomic, retain) IBOutlet UISegmentedControl
*capturedToggle;
@property (nonatomic, retain) IBOutlet UILabel
*capturedDateLabel;
```

这要和其他@properties在一起。

FugitiveDetailViewController.h

【拼吾爱-最新编程资源的分享下载站】

```
@synthesize fugitive, fugitiveNameLabel, fugitiveIdLabel,
fugitiveDescriptionView, fugitiveBountyLabel, capturedDateLabel,
capturedToggle;
```

FugitiveDetailViewController.m

```
-(void) viewWillAppear:(BOOL)animated {
      [super viewWillAppear:animated];

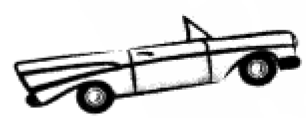
      fugitiveNameLabel.text = fugitive.name;
      fugitiveIdLabel.text = [fugitive.fugitiveID stringValue];
      fugitiveDescriptionView.text = fugitive.desc;
      fugitiveBountyLabel.text = [fugitive.bounty stringValue];
      capturedDateLabel.text = [fugitive.captdate description];
      capturedToggle.selectedSegmentIndex = [fugitive.captured
boolValue] ? 0 : 1;
 }
```

将日期转换成描述信息赋给标签。

根据逃犯是否被捕来设置selectedSegmentIndex：
0 = YES，1 = NO。

```
- (void)dealloc {
      [fugitive release];
      [fugitiveNameLabel release];
      [fugitiveIdLabel release];
      [fugitiveDescriptionView release];
      [fugitiveBountyLabel release];
      [capturedDateLabel release];
      [capturedToggle release];
    [super dealloc];
 }
```

FugitiveDetailViewController.m

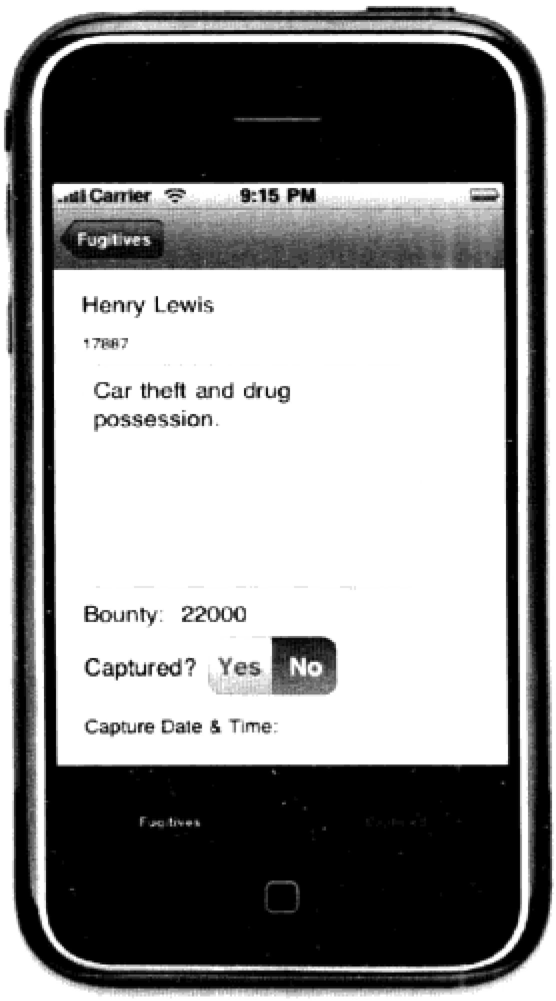


运行测试

构建并调试应用，确保界面工作正常。

【拼吾爱-最新编程资源的分享下载站】

运行测试



所有视图元素看起来都很好！现在我们只要实现它们的行为就可以了……

有问必答

问： 为什么我们不用开关而用分段控件呢？

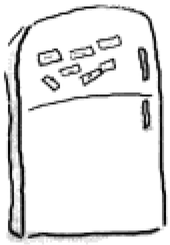
答： 因为没有一种苹果公司批准的方法可以改变开关上的文字。默认情况下，上面的选项是On和Off，这对我们不合用。

问： 为什么不用点选框做被捕标志字段？

答： 那是因为点选框不是一种标准控件。这确实很让人惊讶，因为你在iPhone应用中会非常频繁地见到它。

不过也可以做出来，只要创建一个自定义按钮加上3幅图像（一个空白框、一个选择框加上一个已选中框），然后在它们中间切换就可以了。

【拼吾爱-最新编程资源的分享下载站】



切换代码魔力贴

现在我们已经将控件按照要求布置到位，接着就需要实际赋予它们行动能力了。用下面的魔力贴，实现处理分段控件进行切换的方法。然后就应该万事俱备，只待我们在Interface Builder中将其与分段控件连接起来了。

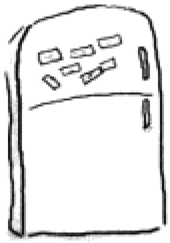
```
- (IBAction) capturedToggleChanged: (id) sender

    if (______________________________     _________) {
        _______  *now = [NSDate  _________
        ___________________ = now;
          fugitive.captured = [NSNumber numberWithBool:YES];
    }
    else {
        ___________________ = nil;
          fugitive.captured = [NSNumber numberWithBool:NO];
    }

    capturedDateLabel.text = [ _________________________ ];
}
```

- x == 1
- fugitive.captdate description
- NSDate
- fugitive.captdate
- fugitive.captdate
- date];
- text
- NSDate
- capturedToggle.selectedSegmentIndex
- x == 0)

【拼吾爱-最新编程资源的分享下载站】



切换代码魔力贴答案

现在我们已经将控件按照要求布置到位，接着就需要实际赋予它们行动能力了。用下面的魔力贴，实现处理分段控件进行切换的方法。然后就应该万事俱备，只待我们在Interface Builder中将其与分段控件连接起来了。

```
- (IBAction) capturedToggleChanged: (id) sender
{
    if ( capturedToggle.selectedSegmentIndex == 0) {
        NSDate *now = [NSDate date];
        fugitive.captdate = now;
        fugitive.captured = [NSNumber numberWithBool:YES];
    }
    else {
        fugitive.captdate = nil;
        fugitive.captured = [NSNumber numberWithBool:NO];
    }
    capturedDateLabel.text = [fugitive.captdate description];
}
```

记住，在我们的控件中分段0是YES。

这只会在控件的值确实发生变化时被调用，所以如果选中下标是0，那么逃犯在调用前就未被抓获。

这会返回一个NSDate，被设置为当前的日期和时间。

Core Data将布尔值存储为NSNumbers，所以我们需要将布尔型的YES/NO转换为NSNumbers，才能修改逃犯信息。

如果逃犯没有被抓获，清除旧的被捕日期（如果有）。

这会返回一段文字，表示了这个NSDate值。

text　NSDate　x == 1

这些是多余的…

按图索骥！

将上面的代码添加到FugtiveDetailViewController.m中，不要忘记在.h文件中做相应的声明：

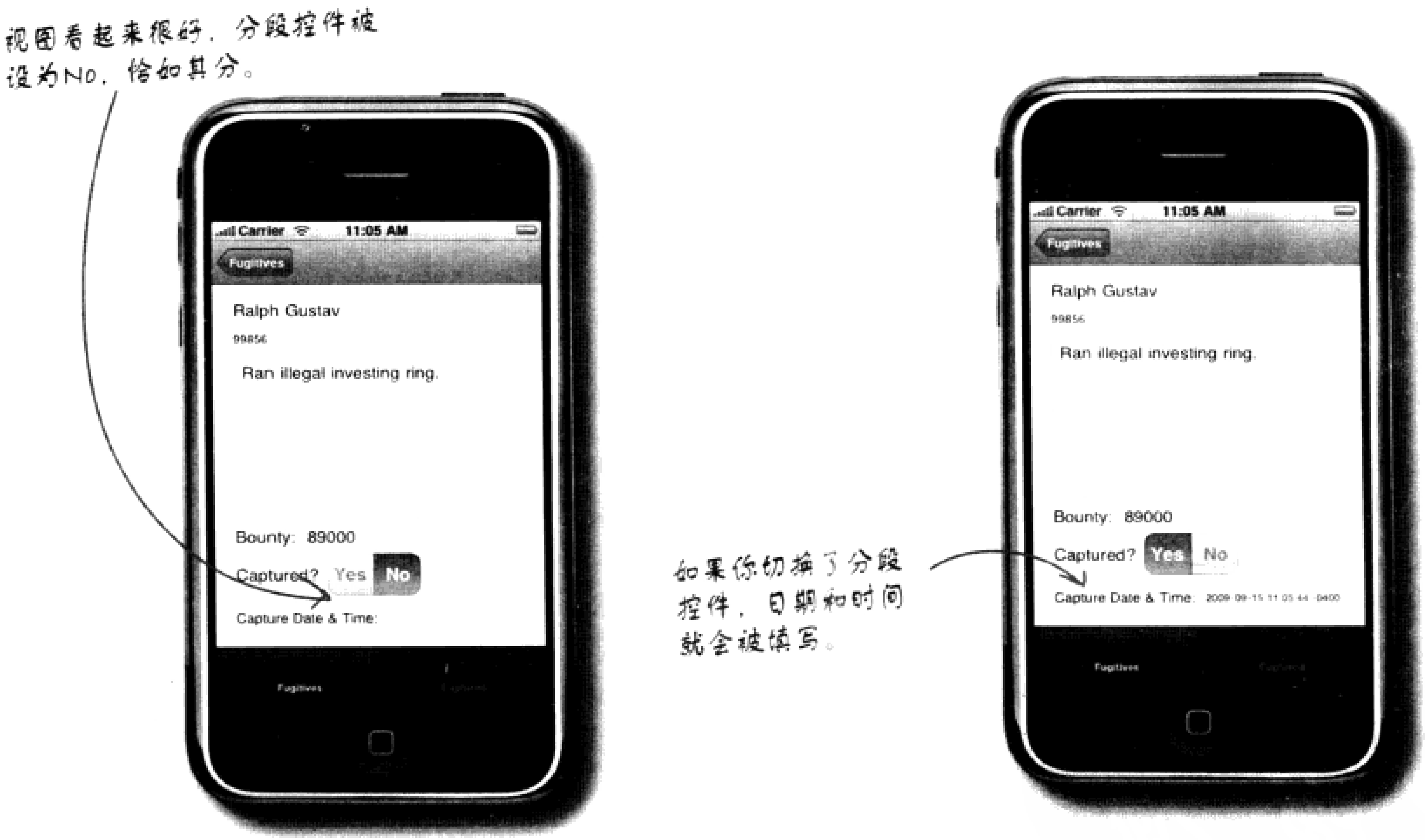
```
- (IBAction) capturedToggleChanged:
(id) sender;
```

最后，在Interface Builder中将分段控件对应的输出口capturedToggle与文件所有者连接起来，还要将来自分段控件的valuechanged事件与文件所有者中的动作方法capturedToggleChanged联系在一起。

【拼吾爱-最新编程资源的分享下载站】

运行测试

现在所有工作都已完成，你应该已经拥有了功能完备的细节视图。那就试一试吧……



工作正常！花点时间，进入表格视图将一个逃犯标成已被捕，然后回到同一个逃犯。继续，请你退出应用并再次检查一遍情况。发生了什么事？

【拼吾爱-最新编程资源的分享下载站】

等一下。我回到表格视图的话，数据还在那里——甚至我在模拟器中完全退出再进入应用，数据也不会丢失。数据被保存了？到底怎么回事？



Core Data同样负责保存！

当你创建应用时选中的那个Core Data框所做的事要超出你的想象——它同样也开启了保存功能。

受控对象上下文会保存新建或者更改的内容

我们已经用受控对象上下文来加载Fugitive了，其实它也负责协调保存数据的过程。记得NSManagedObject是怎样跟踪实体的变化的吗？受控对象上下文可以利用这些信息来分辨它自己管理的对象是否有变化。类似地，如果你创建了一个新的NSManagedObject实例，就需要告诉该实例它属于哪一个受控对象上下文，而该上下文也会知道有新实体需要自己进行状态跟踪。Core Data模板利用这一点，在应用程序退出时查看受控对象上下文中是否有新建或改变的数据。如果有，应用程序就会请求上下文保存这些数据。

这段代码来自iBountyHunterAppDelegate.m，在退出应用时检查是否有数据被修改。

```
- (void)applicationWillTerminate:(UIApplication *)
application {


    NSError *error = nil;
    if (managedObjectContext != nil) {
        if ([managedObjectContext hasChanges] &&
![managedObjectContext save:&error])
    ...
```

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：你们说，如果我创建了NSManagedObjects的新实例，就要告诉它属于哪个受控对象上下文。我该怎么做到这一点呢？

答： 这是我们在第7章提到过的EntityDescription的一部分。如果你要创建一个NSManagedObject的新实例，只要这样做就可以了：[NSEntityDescription insertNewObjectForEntityForName:@"Fugitive" inManagedObjectContext:managedObjectContext];。受控对象上下文从一开始就提供给实例了。

问：传给保存调用的那个"&error"参数是干什么用的？

答： 大多数Core Data的加载/保存操作发生错误时都会有一个NSError指针。Objective-C里的"&"行为与C或C++中一样，会返回对象的"地址"。我们声明一个NSError的指针，然后将这个指针的地址传进保存方法中，以防万一发生什么意外。如果保存调用失败了，Core Data会将更详细的信息填到这个错误参数中。

问：说到错误，如果这个参数带回来一个错误我该怎么办？

答： 这完全依赖应用程序。根据你检测到错误的时机不同，你可以对用户发出警告并试着从错误中恢复；其他一些时候你能做的则很有限。例如，如果错误发生在applicationWillTerminate方法中，你基本没什么可以做的，只能通知用户保存失败，可能还会在其他什么地方把数据保藏起来。

问：我是不是应该只在applicationWillTerminate里调用保存方法呢？

答： 不，根本不是。Core Data模板这样设置只是为了方便，但是你的应用程序应该一有时机就进行保存。实际上，如果你用SQLite数据库作为数据的后端，保存起来要比我们在DrinkMixer中用plist快得多。你应该在对数据做出添加或者修改后尽快进行保存，以避免发生任何形式的数据丢失。

问：你们说过Core Data可以进行数据校验；到底在哪个步骤进行呢？

答： 最起码，Core Data会在将对象存入持久存储之前对其进行校验。所以，如果在受控对象之中有无效数据的话，你有可能在试图保存修改时才收到一个校验错误。为了避免发生这种通知延迟，你应该在修改发生后尽早对你的NSManagedObjects进行校验。你可以以如下方式显式地校验一个新建的NSManagedObject：[fugitive validateForInsert:&error]。与之类似，也有方法对更新和删除进行验证。你随时可以调用这些方法来验证NSManagedObject是否满足你在数据模型中提出的约束条件。如果不满足，你可以通知用户并要求他们修正问题。

问：如果我的修改不想在受控对象上下文中保存了该怎么办？我能将修改复原吗？

答： 比这还要简单——只要向它发送一个rollback:消息就行。当受控对象上下文被要求回滚时，会废弃任何新插入对象、任何删除操作以及任何对已有对象的未保存修改。你可以将受控对象上下文想象成对事务的管理——对实体的修改，包括插入和删除，要么通过一个save:消息提交，要么通过一个rollback:消息放弃。

【拼吾爱-最新编程资源的分享下载站】

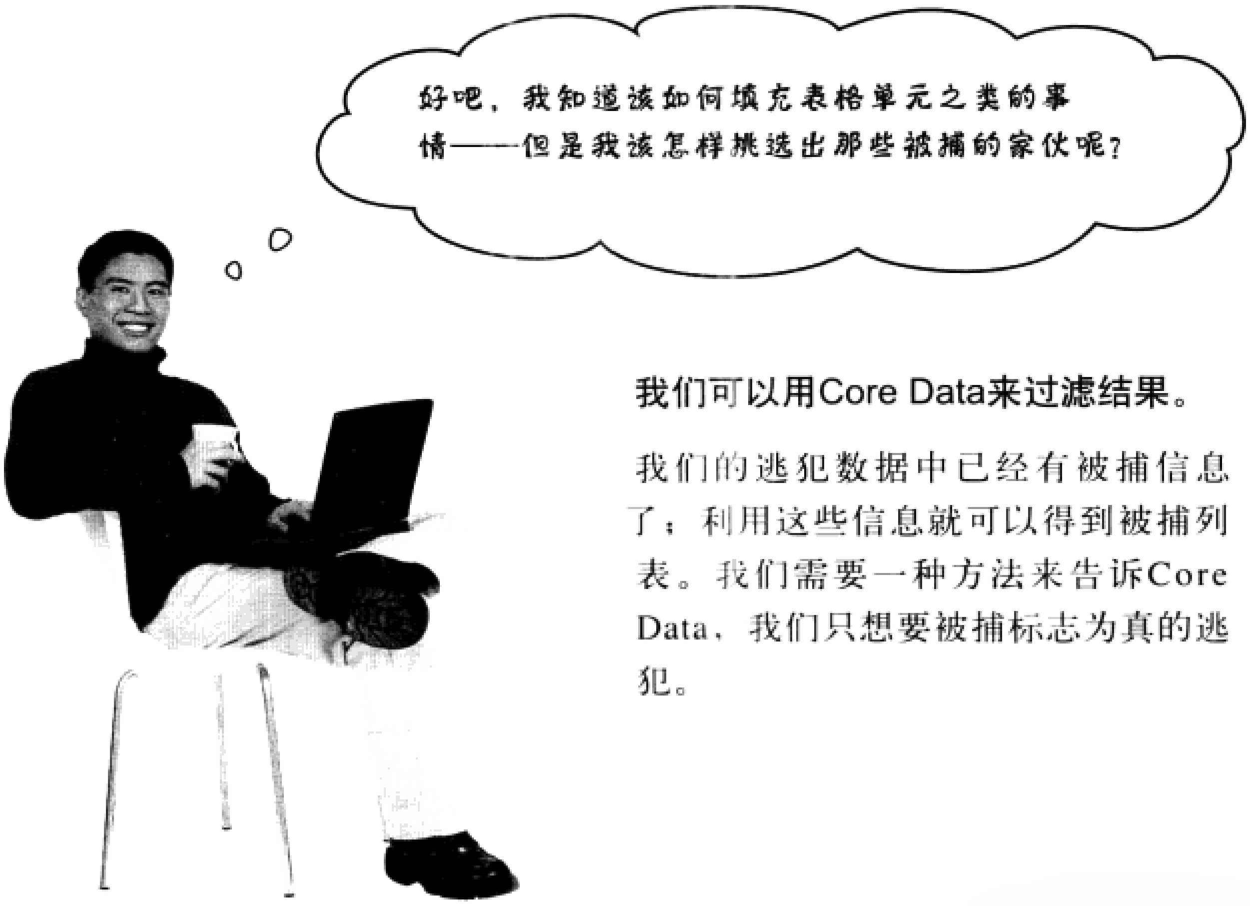
为鲍勃做一个快速演示

看到细节视图和所有抓获罪犯的内容之后，鲍勃被震惊了，但是马上有了一点意见：



做了这么多事，我们却忘记填充被捕列表了！

【拼吾爱-最新编程资源的分享下载站】



我们可以用Core Data来过滤结果。

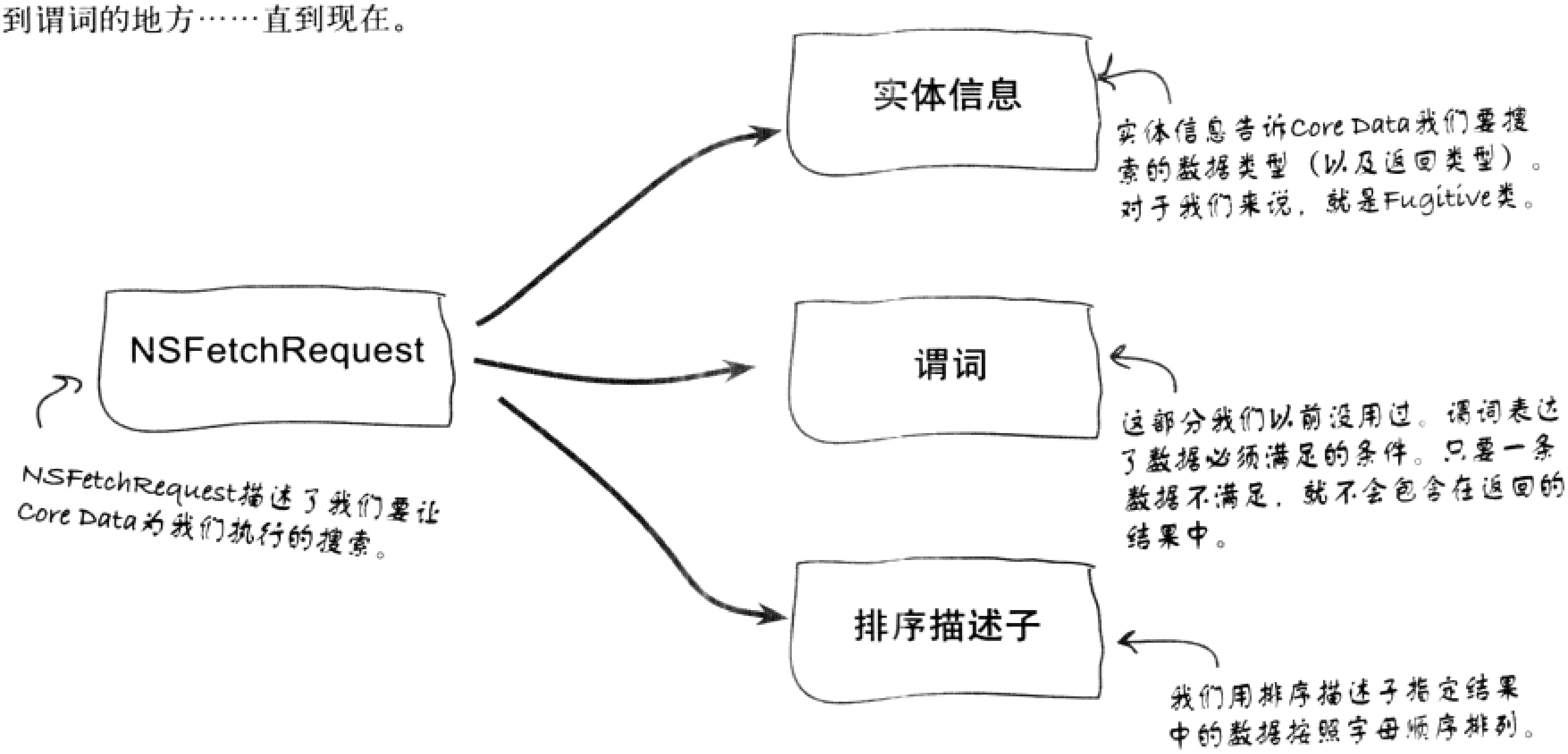
我们的逃犯数据中已经有被捕信息了；利用这些信息就可以得到被捕列表。我们需要一种方法来告诉Core Data，我们只想要被捕标志为真的逃犯。

在哪里放置这种过滤条件合适呢？

【拼吾爱-最新编程资源的分享下载站】

用谓词来过滤数据

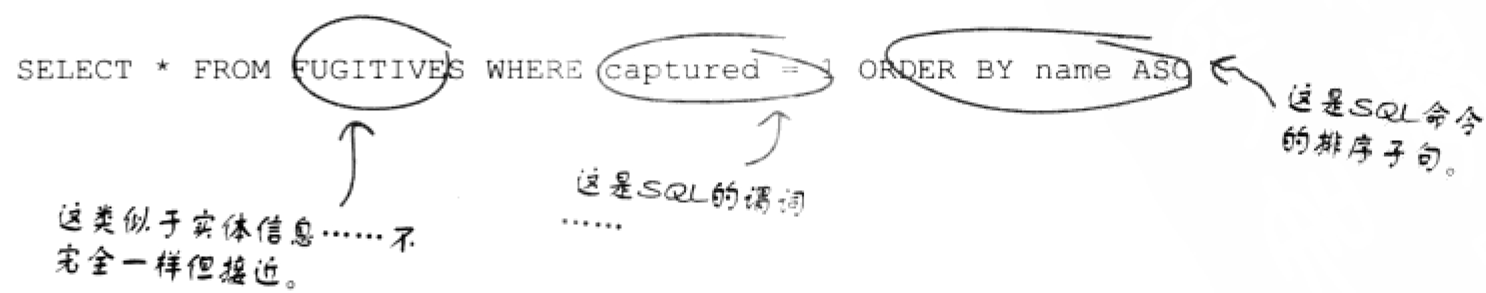
在全世界的数据库语言中，谓词都用来对搜索进行范围限定，以便只找到符合特定条件的数据。还记得在第7章中我们讲过的NSFetchRequest吗？我们已经用到了实体信息和排序描述子，但是还没有要用到谓词的地方……直到现在。



NSFetchRequest中的概念几乎与SQL完全一样

SQL是一种管理数据库的语言。

NSFetchRequest中的3个主要概念几乎与标准SQL中的表达式完全一样：



我们要做的，不过就是为NSFetchRequest提供谓词信息，然后让Core Data处理剩下的问题。我们可以用一个NSPredicate来做这件事……

【拼吾爱-最新编程资源的分享下载站】

我们需要在NSFetchRequest上设置一个谓词

NSPredicate这个类看起来很简单，这一点颇具迷惑性。它能让我们在NSFetchRequest上表达逻辑约束条件。你可以用实体和属性名称辅以比较运算符，表达你的约束信息。你可以用一个由NSString字符串表示的语法来创建基本的NSPredicate，如下所示：

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"captured == YES"];
[request setPredicate:predicate];
```

但是NSPredicate可不仅仅是简单的属性比较。苹果公司提供了几个子类，比如NSComparisonPredicate、NSCompoundPredicate和NSExpression，以及一种复杂的语法，可以进行通配符比较、对象图遍历等。对于iBountyHunter，一个简单的属性条件就足以满足我们的需要，可以让鲍勃的视图工作起来。

练习

是时候填充被捕视图了！要做几项工作，才能让被捕视图变得像如今的逃犯视图一样，然后还要做一点改动，按照我们的需要显示视图。

1. **设置一些被抓获的逃犯。**
在做任何修改之前，先构建并运行旧版本的应用，将一些逃犯的状态切换成已被捕。我们需要这些数据来进行测试。

2. **将被捕视图改得像逃犯视图一样。**
这是我们在第7章中落下的，我们也还没有完成填充被捕列表的工作。由于只是想过滤逃犯列表中的数据，所以最简单的方法就是先使用完整列表，然后再加上过滤代码。别忘了表格视图数据源和代理方法。

3. **添加谓词代码。**
修改NSFetchRequest，让它用一个NSPredicate来限制只查找被捕的逃犯。这需要修改CapturedViewController.m中的viewWillAppear方法。

【拼吾爱-最新编程资源的分享下载站】



练习答案

你应该认得这些来自第7章的代码，它们可以让被捕视图工作起来，然后用谓词代码就可以得到过滤后的数据。

1. 设置一些被抓获的逃犯。 ← 你可以搞定这步——随便挑5个吧！

2. 将被捕视图改得像逃犯视图一样。

```
@interface CapturedListViewController : UITableViewController {
      NSMutableArray *items;
}

@property (nonatomic, retain) NSMutableArray *items;

@end
```



CapturedListViewController.h

```
#import "CapturedListViewController.h"
#import "iBountyHunterAppDelegate.h"
#import "Fugitive.h"
#import "FugitiveDetailViewController.h"

@implementation CapturedListViewController
@synthesize items;
```



CapturedListViewController.m

【拼吾爱-最新编程资源的分享下载站】


```
- (void)viewWillAppear: (BOOL) animated {
    [super viewWillAppear:animated];
      iBountyHunterAppDelegate *appDelegate = (iBountyHunterAppDelegate*)
[[UIApplication sharedApplication] delegate];
      NSManagedObjectContext *managedObjectContext = appDelegate.
managedObjectContext;
      NSFetchRequest *request = [[NSFetchRequest alloc] init];
      NSEntityDescription *entity = [NSEntityDescription
entityForName:@"Fugitive" inManagedObjectContext:managedObjectContext];
      [request setEntity:entity];
      NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
initWithKey:@"name" ascending:YES];
      NSArray *sortDescriptors = [[NSArray alloc]
initWithObjects:sortDescriptor, nil];
      [request setSortDescriptors:sortDescriptors];
      [sortDescriptors release];
      [sortDescriptor release];

      NSError *error;
      NSMutableArray *mutableFetchResults = [[managedObjectContext
executeFetchRequest:request error:&error] mutableCopy];
      if (mutableFetchResults == nil) {
            // 进行错误处理。
      }

      self.items = mutableFetchResults;
      [mutableFetchResults release];
      [request release];
      }
```

这段代码和我们用于FugitiveListViewController的完全一样。

CapturedListViewController.m

【拼吾爱-最新编程资源的分享下载站】



练习答案

你应该认得这些来自第7章的代码，它们可以让被捕视图工作起来，然后用谓词代码就可以得到过滤后的数据。

2 将被捕视图改得象逃犯视图一样（续）。

```
#pragma mark Table view methods

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
      return 1;
}
// Customize the number of rows in the table view.
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NS
Integer)section {
      return [items count];
}
// Customize the appearance of table view cells.
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndex
Path:(NSIndexPath *)indexPath {
      static NSString *CellIdentifier = @"Cell";
      UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifie
r:CellIdentifier];
      if (cell == nil) {
      cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyle
Default reuseIdentifier:CellIdentifier] autorelease];
      }
      // Set up the cell...
      Fugitive *fugitive = [items objectAtIndex:indexPath.row];
      cell.textLabel.text = fugitive.name;
      return cell;
  }
```

CapturedListViewController.m

【拼吾爱-最新编程资源的分享下载站】


```
- (void)tableView:(UITableView*)tableView didSelectRowAtIndexPath:(NSI
ndexPath *)indexPath {
      FugitiveDetailViewController *fugitiveDetailViewController =
[[FugitiveDetailViewController alloc] initWithNibName:@"FugitiveDetail
ViewController" bundle:nil];
      fugitiveDetailViewController.fugitive = [self.items
objectAtIndex:indexPath.row];
      [self.navigationController pushViewController:fugitiveDetailVie
wController animated:YES];
      [fugitiveDetailViewController release];
}

- (void)dealloc {
      [items release];
      [super dealloc];
@end
```

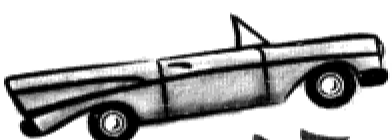
CapturedListViewController.m

3 添加谓词代码。

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"captured ==
YES"];
[request setPredicate:predicate];
```

把这段代码放在viewWillAppear中，紧接在[request setEntity:entity];后面。

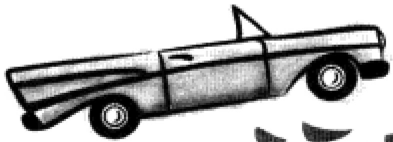
CapturedListViewController.m



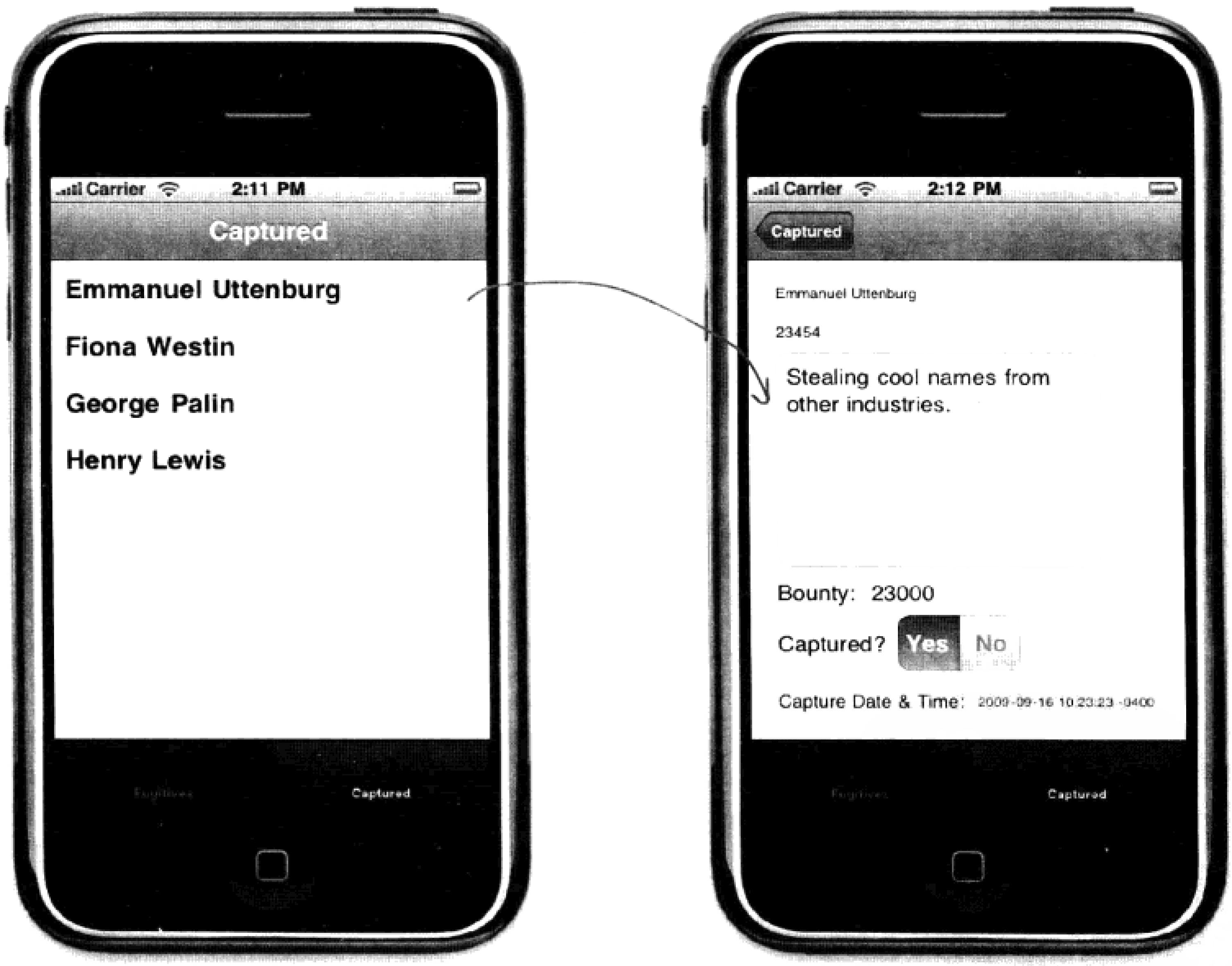
运行测试

继续开动马力——被捕视图应该已经准备好出发了！

【拼吾爱-最新编程资源的分享下载站】



运行测试



运行起来了！这就是我们标成被捕的那四个逃犯。

【拼吾爱-最新编程资源的分享下载站】



是的，我们可以让这些变得更加高效。

但不是把它挪到viewDidLoad里。如果我们把代码挪到那儿，就会出现两个新问题。我们需要另外一种解决方案……

脑力举重

如果我们把获取数据的代码挪到viewDidLoad里会引入什么问题？为了提高性能，我们还可以怎样做？

【拼吾爱-最新编程资源的分享下载站】

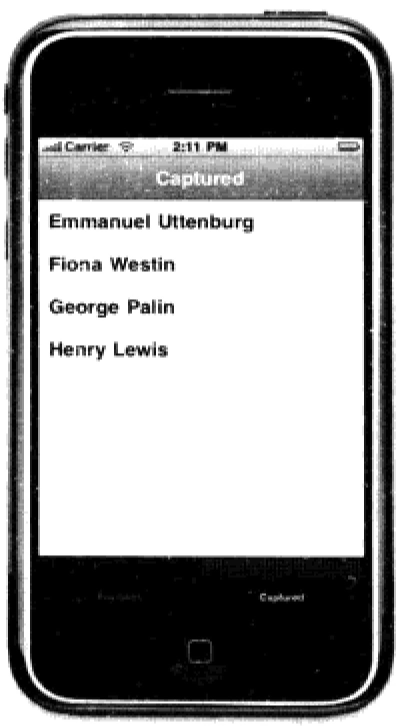
Core Data控制器类提供了高效的结果处理

FugitiveListViewController和CapturedListViewController的代码都在viewWillAppear里。问题是viewWillAppear在每次视图显示的时候都会被调用，也就意味着我们每次都要重新载入所有逃犯和所有被捕逃犯，而不管数据有没有发生过变化。

我们可以把这段代码挪到viewDidLoad中，但是这样的话就只有视图从nib中加载的时候才会调用了。这样会导致两个问题。首先，如果我们将一个逃犯标成了被捕状态，被捕列表不会反映出这一变化，因为它只加载一次数据。第二个问题是，viewDidLoad会在applicationDidFinishLaunching之前被调用，这就是说在应用代理有机会将主数据库复制到位之前，视图就会尝试获取数据。我们需要一种更好的方法来管理数据获取。

表格视图和NSFetchedResultsControllers是天生一对

由于UITableView这种组件太常用了，经常用于处理大量数据，所以专门有一个Core Data类被设计用于支持它。NSFetchedResultsController和受控对象上下文配合工作，这样NSFetchRequest会赋予你一些相当炫的能力：



非常高效的内存使用

NSFetchedResultsController与NSFetchRequest以及受控对象模型一起工作，将实际存留在内存中的数据量最小化。比如，即使我们有10000个逃犯要处理，NSFetchedResultsController也会试着只将UITableView需要显示的那些留在内存中，大多数情况下是将近10到15个。

高性能的UITableView支持

UITableView需要知道自己有多少段落，每个段落中又有多少行，等等。NSFetchedResultsController对此功能有内建支持，不用加载所有数据，就可以快速找出这些信息。

内建支持数据变化监视功能

我们已经讲过，当数据被修改时，受控对象上下文是如何得知这一情况的。NSFetchedResultsController可以利用这一功能，当符合你的搜索结果的数据被修改时，让你（好吧，其实是它的代理）得知此事。

【拼吾爱-最新编程资源的分享下载站】

打造高效的“流线型”代码

我们需要进行少量的重构，以便用上NSFetchedResultsController，但是一旦完工，鲍勃就算给我们一个有100 000个逃犯的数据库，iBountyHunter也可以不费吹灰之力跑起来。我们会把这些放在CapturedListViewController里，但是对FugitiveListViewController也可以进行同样的重构。

首先，我们需要用一个NSFetchedResultsController实例来替换内容数组，如下所示：

我们要让控制器来告诉我们数据什么时候发生变化。

```
@interface CapturedListViewController : UITableViewController
<NSFetchedResultsControllerDelegate> {
      NSFetchedResultsController *resultsController;
}
@property (nonatomic, retain) NSFetchedResultsController
*resultsController;
@end
```

CapturedListViewController.h

去掉内容数组和它的属性。我们已经不需要这些了。

```
@implementation CapturedListViewController
@synthesize resultsController;
```

```
- (void)dealloc {
    [resultsController release];
    [super dealloc];
}
@end
```

删除内容数组相关的代码，改为释放新的结果控制器。

CapturedListViewController.m

下面我们需要修改查询，让它使用这个控制器……

【拼吾爱-最新编程资源的分享下载站】

重构viewWillAppear使用结果控制器

```
- (void) viewWillAppear:(BOOL)animated {
        [super viewWillAppear:animated];

        if (self.resultsController != nil) {
                return;
        }
        iBountyHunterAppDelegate *appDelegate = (iBountyHunterAppDelegate*)
[[UIApplication sharedApplication] delegate];
        NSManagedObjectContext *managedObjectContext = appDelegate.
managedObjectContext;
        NSFetchRequest *request = [[NSFetchRequest alloc] init];
        NSEntityDescription *entity = [NSEntityDescription entityForName:@"Fugitive"
inManagedObjectContext:managedObjectContext];
        [request setEntity:entity];

        NSPredicate *predicate = [NSPredicate predicateWithFormat:@"captured ==
YES"];
        [request setPredicate:predicate];

        NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
initWithKey:@"name" ascending:YES];
        NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:sortDescriptor,
nil];
        [request setSortDescriptors:sortDescriptors];
        [sortDescriptors release];
        [sortDescriptor release];

        NSFetchedResultsController *fetchedResultsController =
[[NSFetchedResultsController alloc] initWithFetchRequest:request
   managedObjectContext:managedObjectContext  sectionNameKeyPath:nil
  cacheName:@"captured_list.cache"];
        fetchedResultsController.delegate = self;
        NSError *error;
        BOOL success = [fetchedResultsController performFetch:&error];
        if (!success) {
                // 进行错误处理。
        }

        self.resultsController = fetchedResultsController;
        [request release];

        [self.tableView reloadData];
```

由于NSFetchedResultsController会在数据发生变化时告诉我们，所以我们只需实际获取一次数据。如果已经完成了（视图被再次显示），我们可以直接退出。

用我们的获取数据请求和受控对象上下文来创建并初始化NSFetchedResultsController。

我们做它的代理，这样数据变化时就会被告知。

现在不用请求受控对象模型来执行数据获取了，我们改为请求控制器。

把控制器保藏起来，这样我们就可以取出数据了。

告诉表格视图我们的数据变了。

CapturedListViewController.m

【拼吾爱-最新编程资源的分享下载站】



是的。

访问被获取数据所需要的一切，NSFetchedResultsController都可以给我们。实际上，它可以更加高效地完成这些工作。

拿起笔来

我们已经给了你设置NSFetchedResultsController的代码。现在你需要修改表格视图的代理和数据源方法，让它们不再使用视图，转而使用控制器。

1 **重构numberOfSectionsInTableView和numberOfRowsInSection，让它们使用控制器。**

`NSFetchedResultsController`有一个属性sections，是一个`NSFetchedResultsSectionInfo`对象的数组。想办法用这个数组来确定有多少段落以及每个段中有多少行。

2 **重构 cellForRowAtIndexPath和didSelectRowAtIndexPath，让它们使用控制器。**

`NSFetchedResultsController`令实现这些方法更加轻松，请利用它的objectAtIndexPath方法来完成工作。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

这就是CapturedListViewController.m中表格方法的最终代码。

```
#pragma mark Table view methods
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [[self.resultsController sections] count];
}

// Customize the number of rows in the table view.
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)
section {
    return [[[self.resultsController sections] objectAtIndex:section]
numberOfObjects];
}

// Customize the appearance of table view cells.
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NS
IndexPath *)indexPath {
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIden
tifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier] autorelease];
        }
// Set up the cell...
    Fugitive *fugitive = [self.resultsController
objectAtIndexPath:indexPath];
    cell.textLabel.text = fugitive.name;
    return cell;
}
```

对于段落数，我们可以直接返回控制器中的段落数目。

也可以用一个遵循NSFetchedResultsSectionInfo协议的对象的指针来完成这一工作。

这里没什么特别的——只是从给定的indexPath处得到一个Fugitive对象。

CapturedListViewController.m

【拼吾爱-最新编程资源的分享下载站】


```
- (void)tableView:(UITableView*)tableView didSelectRowAtIndexPath:(NSIndexPath *)
indexPath {

    FugitiveDetailViewController *fugitiveDetailViewController =
[[FugitiveDetailViewController alloc] initWithNibName:@"FugitiveDetailViewControl
ler" bundle:nil];

fugitiveDetailViewController.fugitive = [self.resultsController
objectAtIndexPath:indexPath];
[self.navigationController pushViewController:fugitiveDetailViewControlle
r animated:YES];
[fugitiveDetailViewController release];
```

再一次用indexPath找到Fugitive对象，然后就搞定了。

CapturedListViewController.m



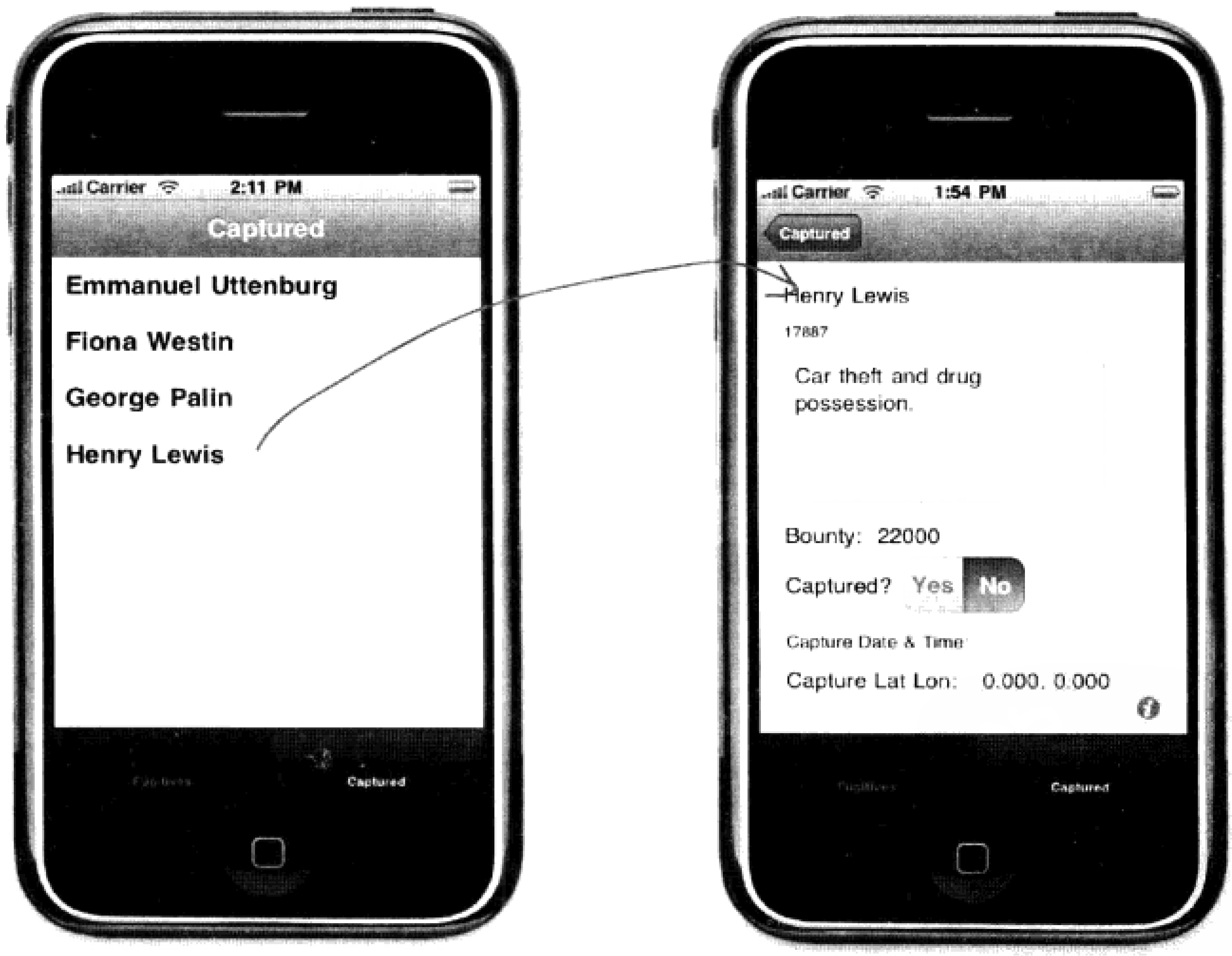
运行测试

运行iBountyHunter，看看改动有没有破坏什么东西。视图应该像以前一样加载……差不多吧。快速进行一些测试——如果你把一个逃犯标成被捕状态，他是不是换到另一个列表中去了？如果用主键退出再进入应用，会怎样？

【拼吾爱-最新编程资源的分享下载站】

运行测试

现在你用的是控制器，而不只是一个谓词了，应用的行为应该是一样的。但是有些家伙即使没有被标成被捕，也出现在被捕列表里了！



脑力练习

为什么逃犯的被捕状态发生变化后，没有正确切换到另一个列表中？

【拼吾爱-最新编程资源的分享下载站】

我们需要刷新数据

当你改变逃犯的状态时，他们不能正确地在列表间切换，这是因为我们没有在每次被捕列表视图被显示时刷新数据。我们需要对NSFetchedResultsController进行设置，让我们能够知道什么时候数据变了，这样才能更新表格。

```
- (void)controllerDidChangeContent:(NSFetchedResultsController *)controller
{
        [self.tableView reloadData];
}
```

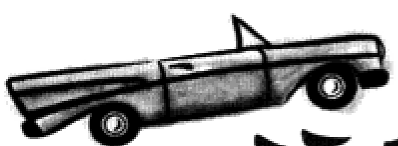
这个方法可以加在CapturedListViewController.m文件中的任何位置。

当发现数据变化时，表格视图会重新加载全部数据。

NSFetchedResultsController可以探测数据变化

现在我们已经将应用设置为使用NSFetchedResultsController，而不只是一个数组，可以利用控制器内嵌的方法帮助我们。视图控制器有内建的功能，可以支持通过代理监视数据的变化。我们已经将自己设置为代理，但还没有实现处理数据变化的代码。

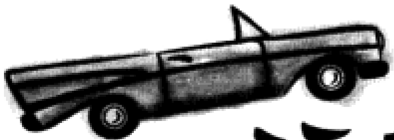
如果你要处理大量的数据，让视图在检测到变化时重新加载全部数据会变得很笨拙；不过FetchedResultsController代理也有内建功能，可以只通知你特定单元格被修改了，这样就可以只改动该单元格。要了解更多细节内容，请查看苹果公司的文档。



运行测试

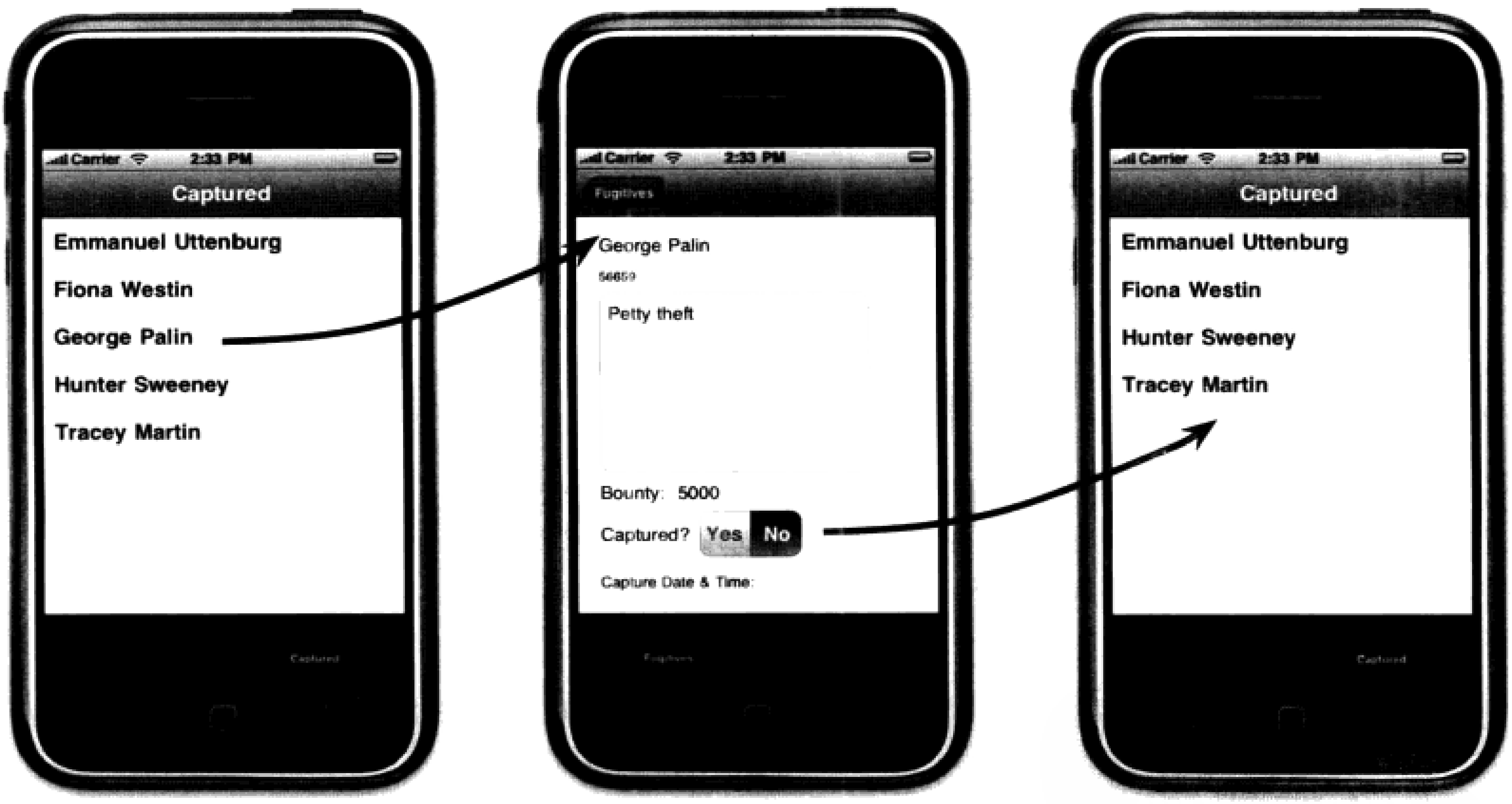
请实现我们在上面列出的controllerDidChangeContent方法，并确认一切工作正常。

【拼吾爱-最新编程资源的分享下载站】



运行测试

要做的事情和上次一样，构建并运行应用，然后改变其中一个逃犯的状态，就可以将他动态地从被捕列表中去掉。



开始有5个被捕的逃犯…

…把一个从列表中移走…

…马上他就消失了！

工作正常！

【拼吾爱-最新编程资源的分享下载站】



没有什么能比一个满意的顾客更让人欣慰了！

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：我在哪儿能找到NSPredicate的全部语法？

答： NSPredicate的语法相当复杂，可以表达对数据的约束条件。在NSPredicate类的文档中对其语法有简单的总结，但是苹果公司还有一份完整的文档，可以帮助你编写高级谓词。

问：看起来，像这样直接将谓词语法输入到代码中很容易发生错误。没有其他的类似嵌入SQL那样的办法吗？

答： 有的，而且Xcode对此可以提供很大的帮助。与将谓词嵌入代码的做法相反，你可以用Xcode的数据建模工具，以图形化的方式进行构建，就像构建受控对象模型一样。要想以图像化的方式构建谓词，请在Xcode中选中一个实体，然后单击加号图标，就像添加一个属性一样。再选择“Add Fetch Request”（添加数据获取请求），就可以创建一个新的数据获取请求了，之后单击标有“Edit Predicate”的按钮，就会出现一个图形化编辑器。你可以随意为数据获取请求命名。想得到这些请求，你得用类似下面这些代码：

```
NSFetchRequest *fetchRequest = [managedObjectModel fetchRequestFromTemplateWithName:@"capturedFugitives" substitutionVariables:[NSDictionary dictionaryWithObject:capturedFlag forKey:@"captured"]];
```

然后，只要用这个获取请求替代原来在代码中创建的那一个就行了。如果你喜欢把它们放在代码里，也可以用Xcode的构建器来组装一个谓词，然后将其复制粘贴到代码中。

问：当数据发生变化时重新载入整个表格看起来很没有效率。我们能不能试着优化一下？

答： 是很没有效率，而且我们确实能进行优化。你还可以实现许多代理方法，能以更细的粒度获得受控对象上下文中的状态信息。有了这些信息，你就可以知道是否仅需更新某个特定单元格，还是需要插入单元格或者删除单元格。我们用了更简便的方法，直接请求表格视图完全重新载入数据。

问：我们传给结果控制器的那个缓存值是干什么用的？

答： 结果控制器会用那个名字的文件对信息进行缓存，比如内容的条目数、段落数等。它会监视数据存储，如果有东西发生变化就会重新生成缓存。你还可以强制它删除缓存，不过一般不会需要这样做的。

问：我们的结果控制器只有一个段落。我怎样才能将内容拆分成多个段呢？

答： 只要为参数sectionNameKeyPath提供一个属性名就行了。NSFetchedResultsController会用这个属性对结果进行分组，将每个组作为一个段落返回。熟练以后，也可以创建一个临时属性来进行分组，这样即使分组标准不是实际存储在数据库中也可以，只要在对象模型中加一个自定义的读取方法，用它计算出临时属性值就可以了。

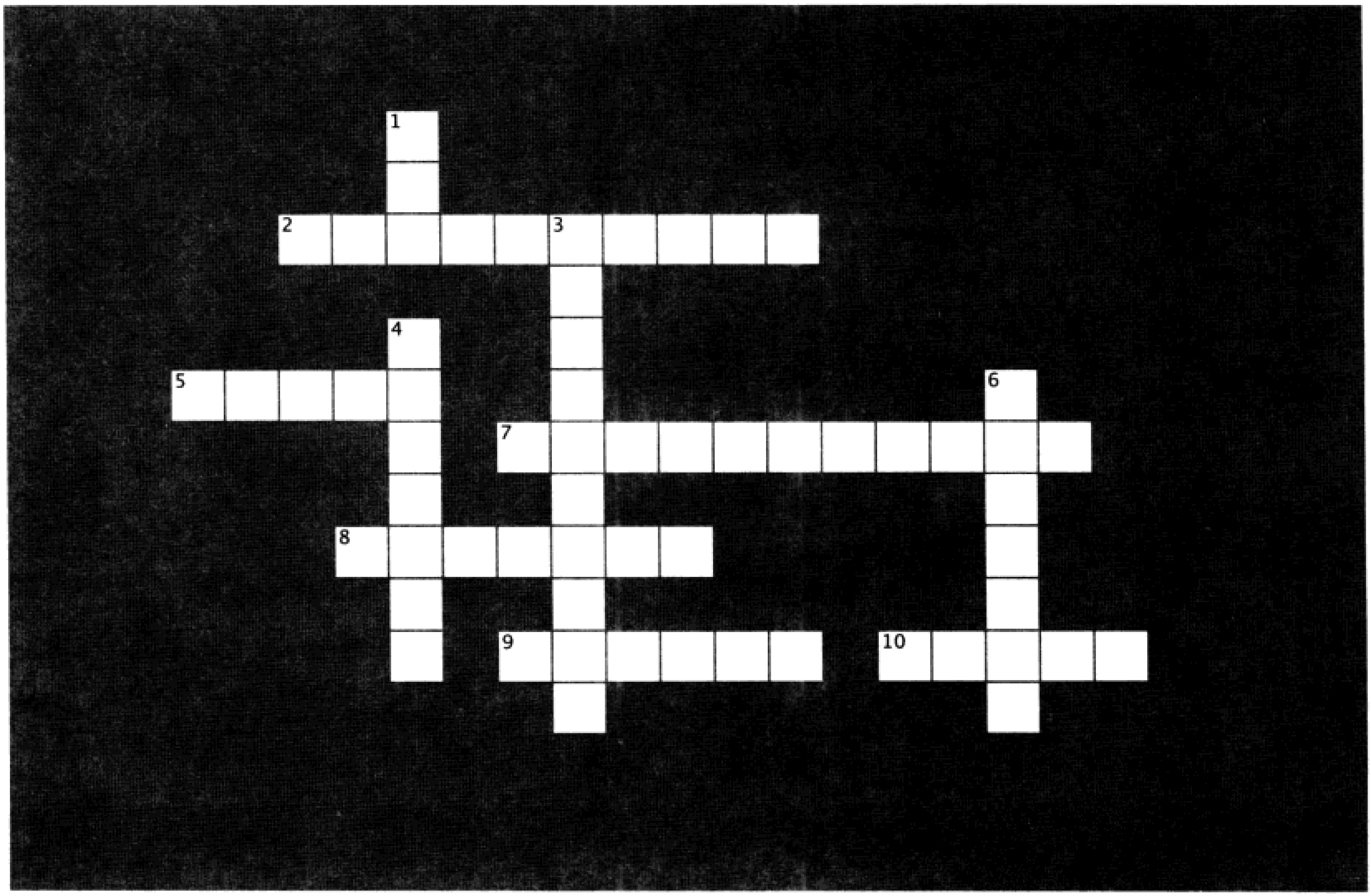
关键要点

- NSFetchRequest可以接受一个NSPredicate，对数据根据逻辑条件进行过滤。
- 你可以在代码中直接表示NSPredicate条件，也可以用Xcode的谓词构建器制作。
- NSFetchedResultsController 有高效的内存管理措施，并为UITableView提供变化监测。
- 把代码放在viewWillAppear中要小心，每次视图显示都会调用它。

【拼吾爱-最新编程资源的分享下载站】

数据迁移纵横字谜

试一试使用这些新的数据行话，展示一下词汇技巧吧……



横向提示

2. viewDidLoad和view__________都会加载视图，但频繁程度不同。

5. 数据_________负责对数据进行读写。

7. 自动迁移被称为__________数据迁移。

8. 要对数据进行升级，我们需要__________数据。

9. NSFetchedResultsController长于进行______管理。

10. NSFetchedResultsController可以__________数据变化。

纵向提示

1. __________中的概念类似于NSFetchRequest中的概念。

3. _________用于过滤数据。

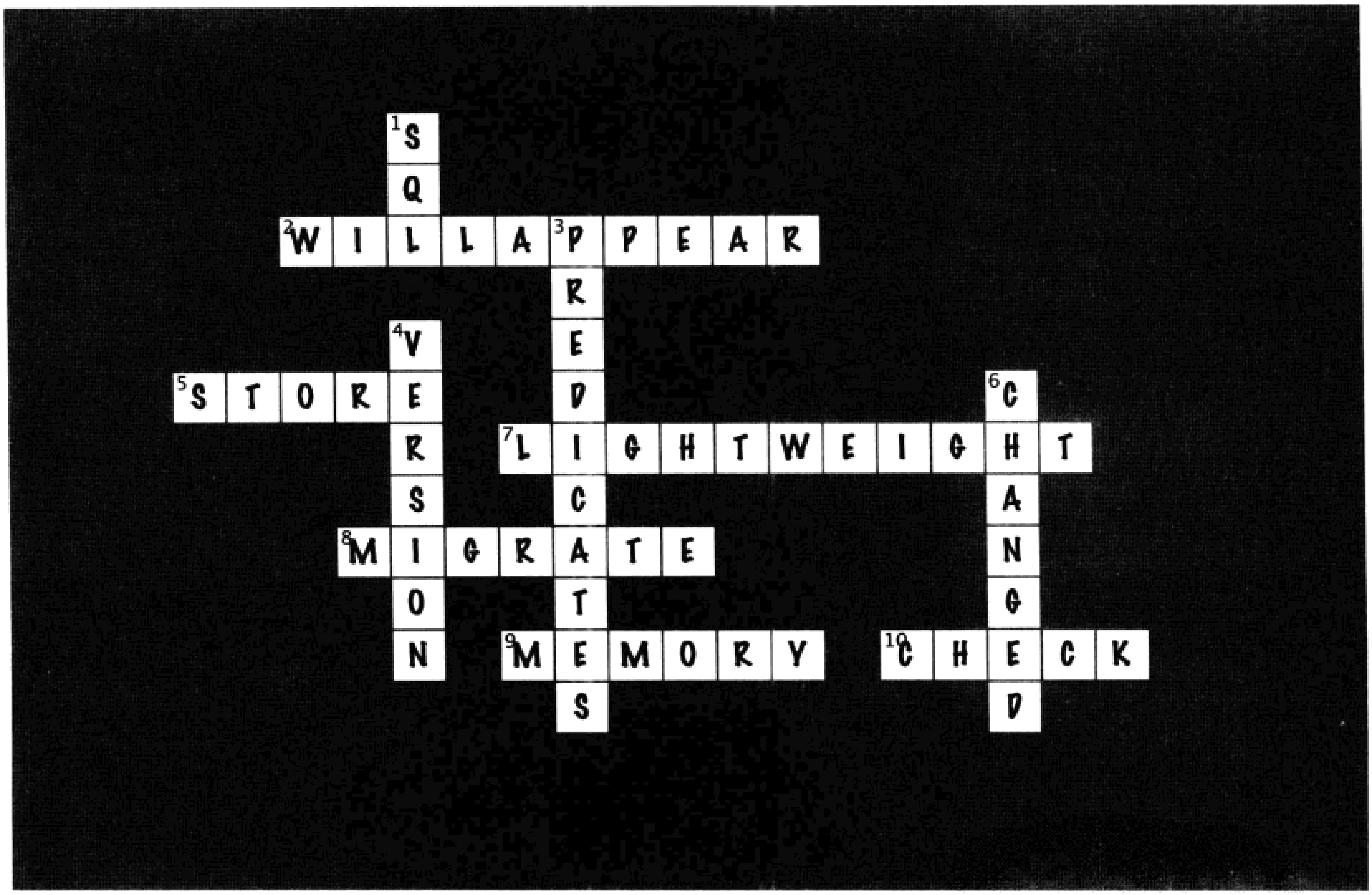
4. 新模型是当前________。

6. 受控对象上下文会保存新建或__________的数据条目。

【拼吾爱-最新编程资源的分享下载站】

数据迁移纵横字谜答案

试一试使用这些新的数据行话，展示一下词汇技巧吧……



横向提示

2. viewDidLoad和view_________都会加载视图，但频繁程度不同。[WILLAPPEAR]

5. 数据________负责对数据进行读写。[STORE]

7. 自动迁移被称为_________数据迁移。[LIGHTWEIGHT]

8. 要对数据进行升级，我们需要_________数据。[MIGRATE]

9. NSFetchedResultsController长于进行______管理。[MEMORY]

10. NSFetchedResultsController可以_________数据变化。[CHECK]

纵向提示

1.____________中的概念类似于NSFetchRequest中的概念。[SQL]

3.__________用于过滤数据。[PREDICATES]

4.新模具是当前___________________。[VERSION]

6.控对象上下文会保存新建或_______的数据条目。[CHANGED]

【拼吾爱-最新编程资源的分享下载站】



你的数据工具箱

你已经消化吸收了第8章，现在把数据迁移与优化加入到你的工具箱中了。如果你想得到一份完整的书中功能提示的列表，请访问http://www.headfirstlabs.com/iphonedev。

数据迁移

Core Data可以用轻量级迁移自动对数据库进行修改。

用版本来控制数据迁移。

轻量级迁移可以用来添加属性或者改变是必需还是可选状态。

持久对象存储

对数据实际进行读写。

可以进行数据迁移，有时甚至不需要实际加载数据就可以完成。

如果模型改变太大，无法进行轻量级迁移，可以使用映射模型。

保存

受控对象上下文会负责保存新建或者修改过的内容。

NSFetch-ResultsController

令内存效率最优化。

对UITableView的高性能支持。

内建支持对数据变化进行监测。

数据过滤

用谓词来对结果数据进行过滤。

谓词需要被设置在NSFetchRequest之上才能起作用。

【拼吾爱-最新编程资源的分享下载站】



iPhone自带一个照相机还真
是不错……

【拼吾爱-最新编程资源的分享下载站】

9 照相机、地图开发包和Core Location

现实世界中的证据

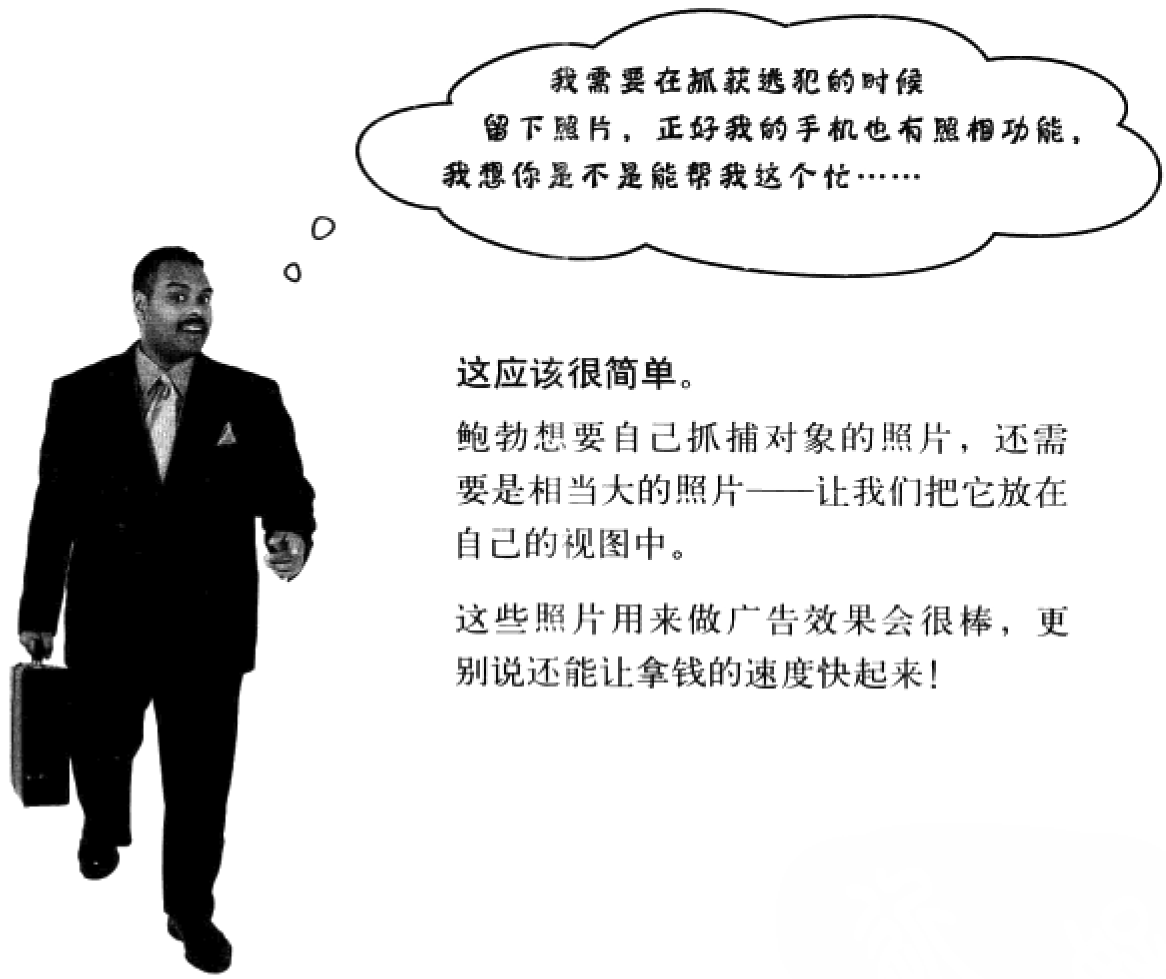


iPhone明晓它身处何方，见到了什么。就像任何一个iPhone用户都知道的那样，iPhone的所作所为远不止管理数据那么简单：它还可以拍摄照片，找出你所在的位置，并整合这些信息让应用来使用。对这些功能进行整合的美妙之处在于，只要点触一下iPhone，进入它提供的工具，导入照片、位置和地图等功能就唾手可得，却根本不用编写多少代码。

【拼吾爱-最新编程资源的分享下载站】

对于鲍勃，收款还需要证据！

鲍勃正在努力工作中，力求将尽可能多的逃犯绳之以法，但是要想收到报酬，他必须把他的捕获对象记录存档。



这应该很简单。

鲍勃想要自己抓捕对象的照片，还需要是相当大的照片——让我们把它放在自己的视图中。

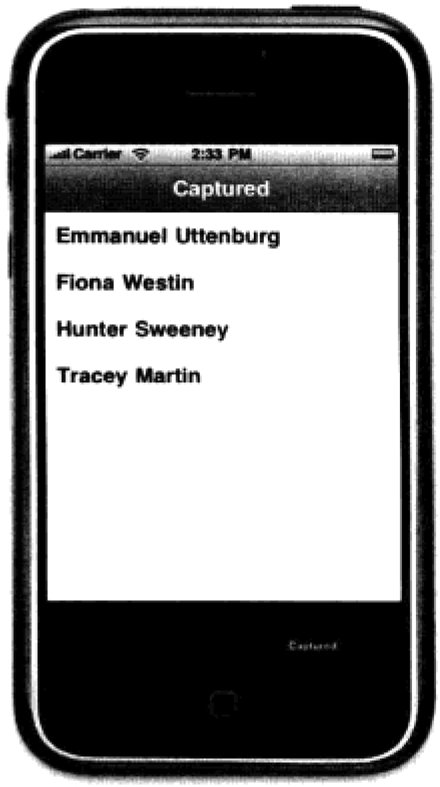
这些照片用来做广告效果会很棒，更别说还能让拿钱的速度快起来！

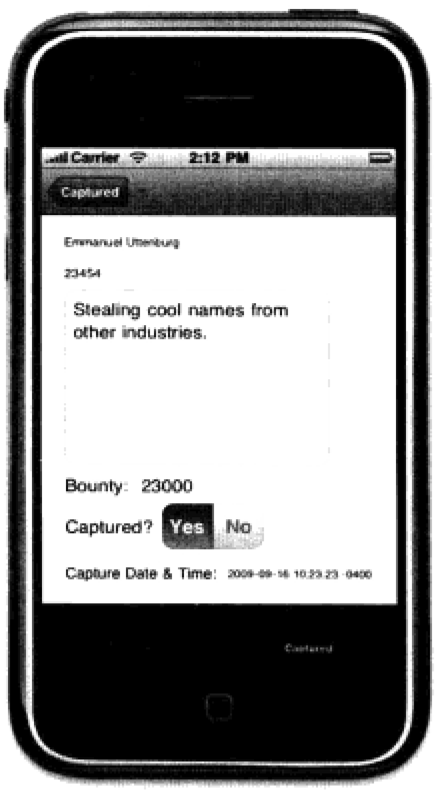
【拼吾爱-最新编程资源的分享下载站】

拿起笔来

这是应用目前看起来的样子。我们该在何处、又该怎样加入对照片的支持呢？





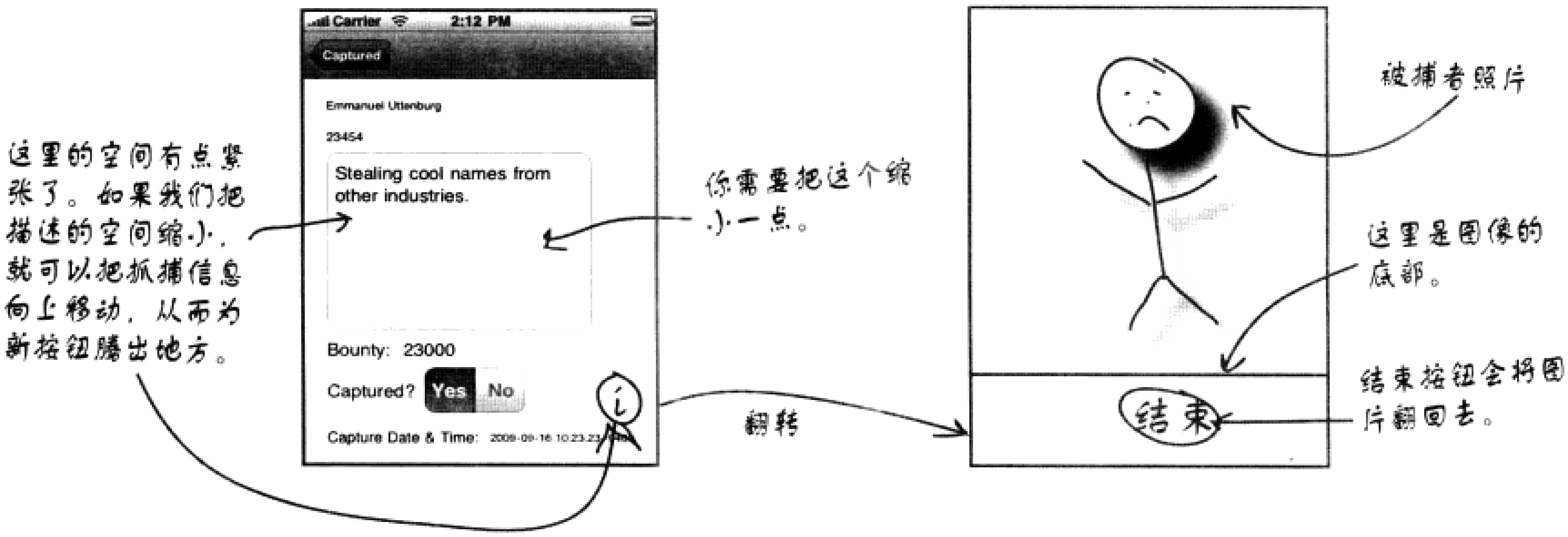


把你的想法画在这里。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

这就是我们弄出来的照片视图。它的工作方式与大多数工具应用类似。



细节视图的翻转功能！

是时候让我们在应用里使用一些真正的动画了。由于我们想让照片在进入细节视图（就是鲍勃用来寻找逃犯的）之后才能见到，所以应该让它粘在细节视图的背面。

这在iPhone上，是一种非常常见的工具应用界面。通常会有两个视图，其中一个视图上面有一个信息按钮，当信息按钮被按下时，就会翻转显示出另一个视图。我们的应用不是工具应用，但是我们可以借鉴这个创意，将我们的逃犯细节视图披上一副好看的棒球卡外衣。

翻转是UIKit自带的又一种转场效果。我们打算用一个模式视图作为前面所说的第二个视图。

作者提示：这些是给下一个练习的提示，请多加留意！

【拼吾爱-最新编程资源的分享下载站】

长篇练习

准备和提示也够多了。请构建出视图并将其实现！

1 首先修改FugitiveDetailViewController。

细节视图需要一个新的信息按钮和一个动作方法，以触发到新视图的翻转动作。信息按钮只是一个常规按钮，类型为“深色信息按钮”。

不要忘了在Interface Builder中将按钮与IBAction连接起来！

2 当信息按钮被按下时，使用自定义的动画来显示新视图。

你已经知道该怎样显示一个模式视图了，但是这次我们要用自定义动画来进行。我们要使用的动画是`UIModalTransitionStyleFlipHorizontal`。如果你对如何使用这个动画感兴趣，请看一下UIViewController的文档。

3 打造新的CapturedPhotoViewController。

这个视图将会同时具有一个UIImageView和一个完成按钮。别忘了将动作方法与按钮绑定在一起来取消这个视图。

先别担心UIImageView的IBOutlet，我们马上就会讲到这个问题。

【拼吾爱-最新编程资源的分享下载站】

长篇练习答案

这次你要添加一大堆功能，但不会有太多外界的帮助！
下面就是我们给你的提示。

1 首先修改FugitiveDetailViewController。

添加这个动作方法，响应按钮的按下动作。

```
- (IBAction) showInfoButtonPressed: (id) sender;
@end
```

FugitiveDetailViewController.h

```
#import "CapturedPhotoViewController.h"
```

FugitiveDetailViewController.m

2 当信息按钮被按下时，使用自定义的动画来显示新视图。

```
- (IBAction)showInfoButtonPressed: (id) sender {

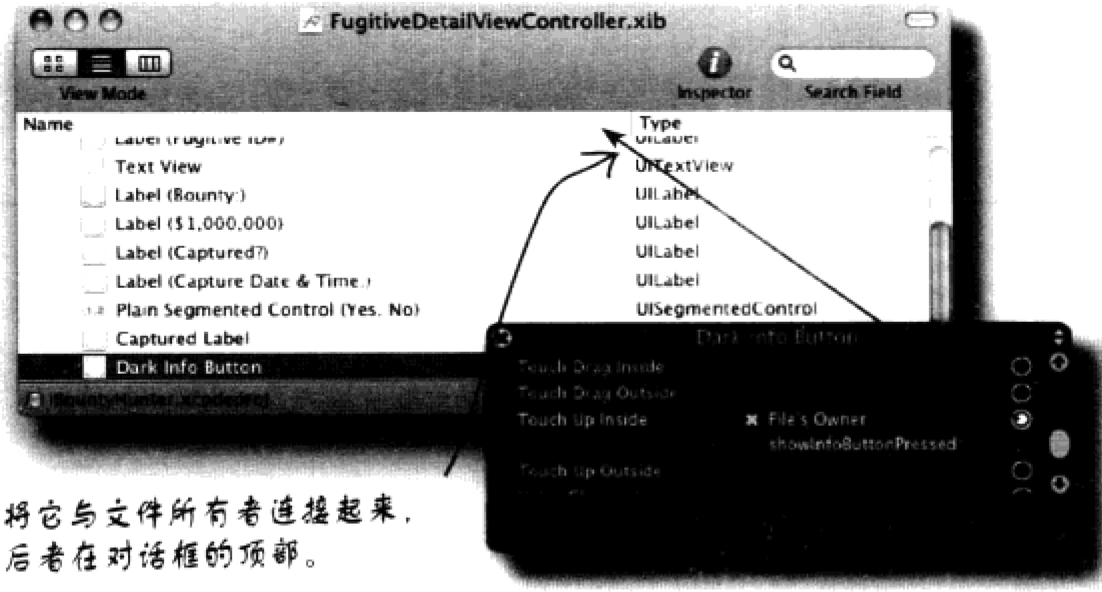
    CapturedPhotoViewController *controller =
      [[CapturedPhotoViewController alloc] initWithNibName:
       @"CapturedPhotoViewController" bundle:nil];

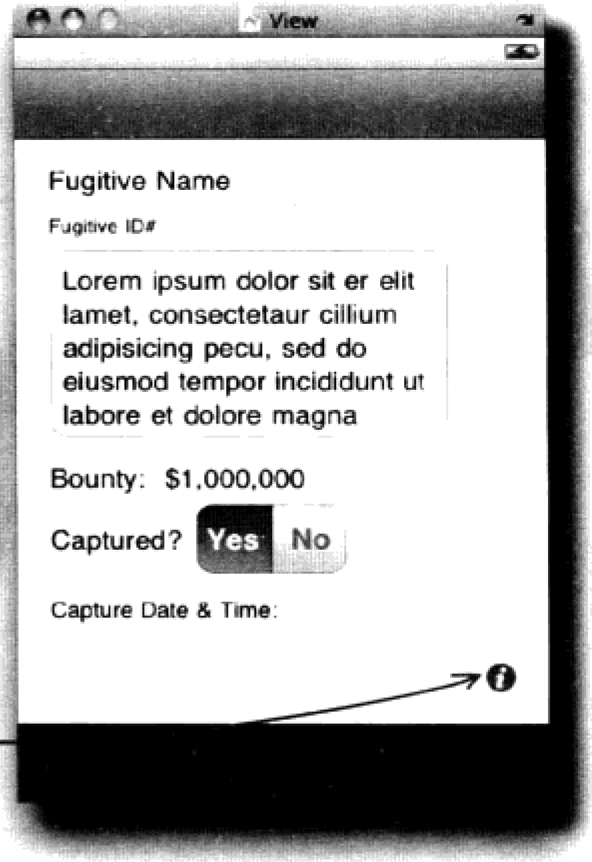
    controller.modalTransitionStyle =
      UIModalTransitionStyleFlipHorizontal;
    [self presentModalViewController:controller animated:YES];
    [controller release];
}
```

实例化视图控制器并打开翻转视图的nib。

FugitiveDetailViewController.m

【拼吾爱-最新编程资源的分享下载站】





将它与文件所有者连接起来，后者在对话框的顶部。

信息按钮只是一个单纯的UIButton，在查看器中被设置为"info dark"类型。请确保这里也有模拟的标签栏，不然按钮就会被挡住了。

3 打造新的CapturedPhotoViewController。

```
@interface CapturedPhotoViewController : UIViewController {
}

- (IBAction) doneButtonPressed: (id) sender;

@end
```

声明视图控制器与按钮的动作方法。

CapturedPhotoViewController.h

```
- (IBAction) doneButtonPressed: (id) sender {
    [self dismissModalViewControllerAnimated:YES];
}
```

当完成按钮被单击时，我们想让视图消失。

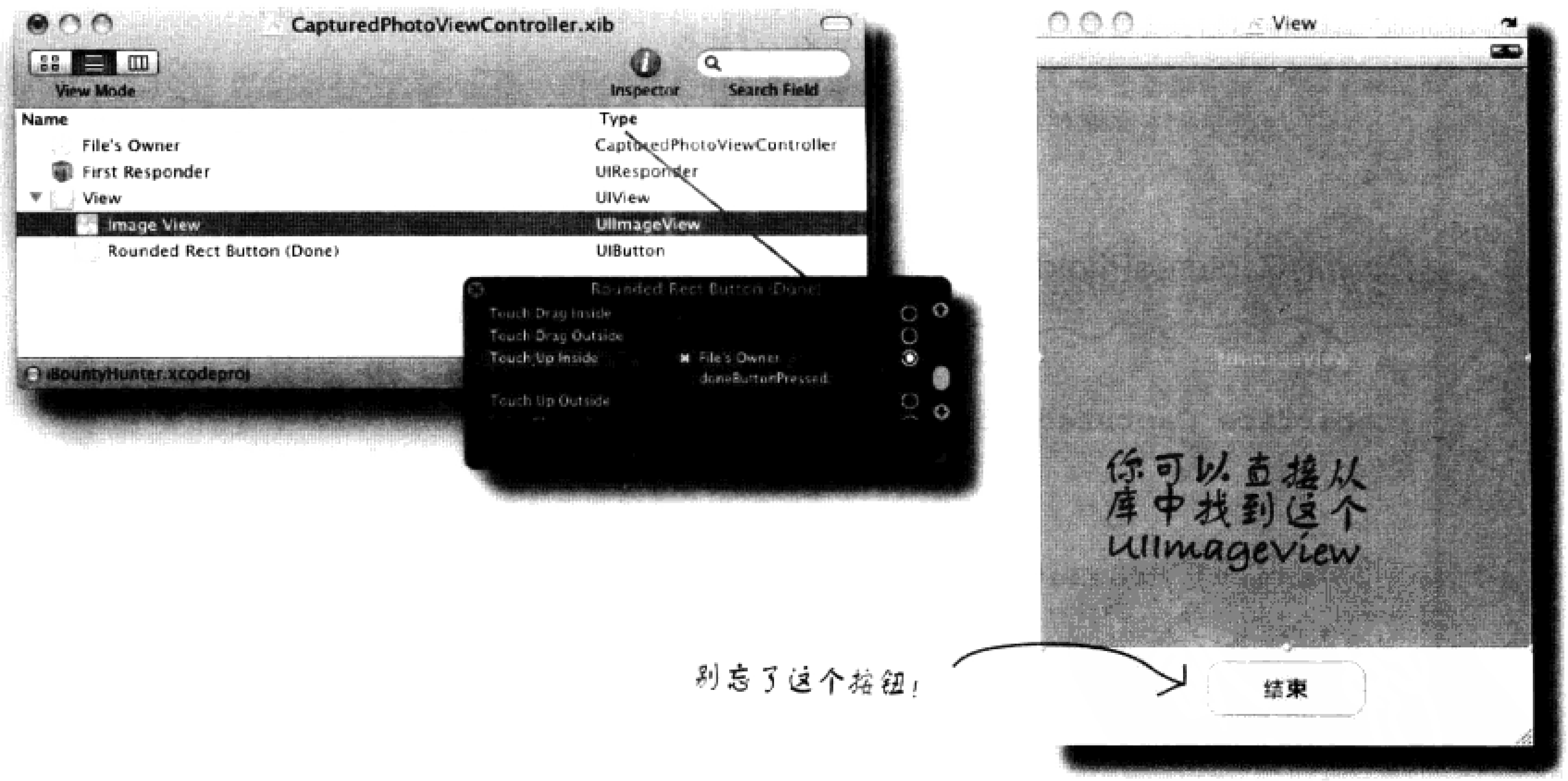
CapturedPhotoViewController.m

【拼吾爱-最新编程资源的分享下载站】

长篇练习答案

这次你要添加一大堆功能，但不会有太多外界的帮助！
下面就是我们给你的提示。

4 打造新的CapturedPhotoViewController(续)。



【拼吾爱-最新编程资源的分享下载站】

运行测试

运行应用，看一看动画工作正常不正常，酷不酷！



拿起笔来

现在视图和动画都工作正常了，图像该怎么办呢？请完成下面的填空，同时考虑一下数据模型。

UIImage应该被储存在 .... 中。

.... 和 .... 需要再次进行 .... ，才能工作。

.... 必须了解图像在哪里，又该在何处显示它。

图像必须来自 .... 或者 .... 。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

现在视图和动画都工作正常了，图像该怎么办呢？
请完成下面的填空，同时考虑一下数据模型。

UIImage应该被储存在 ......数据库...... 中。

......数据库...... 和 ......数据模型...... 需要再次进行 ......迁移...... ，才能工作。

......CapturedPhotoViewController...... 必须了解图像在哪里，又该在何处显示它。

图像必须来自 ......照相机...... 或者 ......相片库......

按图索骥！

以前你已经进行过数据库迁移，这次又要再来一次了。你已经了解了这个过程的来龙去脉，让我们进入Xcode中，再做一次数据库迁移吧。

1. **选中iBountyHunter 2.xcdatamodel。**
   然后在菜单中依次选择**Design→Data Model→Add Model Version**选项。在iBountyHunter.xcdatamodel目录下就会出现一个iBountyHunter 3.xcdatamodel。

2. **设置当前版本。**
   在iBountyHunter.xcdatamodeld目录中，选择iBountyHunter 3.xcdatamodel，这将是我们的新版本模型。在菜单中依次选择**Design → Data Model → Set Current Version**。

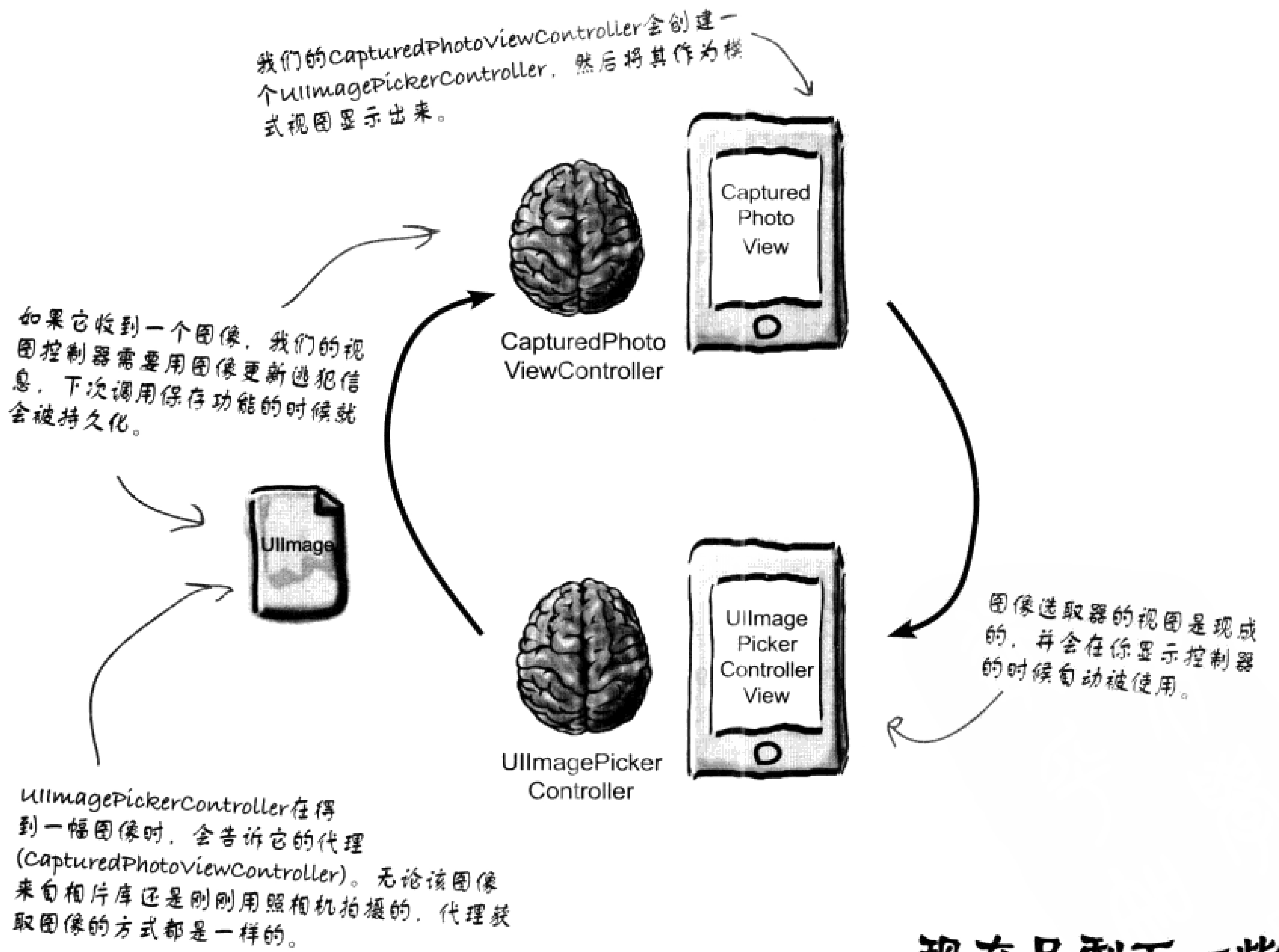
3. **在新的数据模型中添加新字段，并生成新的Fugitive类。**
   对于图像，我们需要用一个新属性，命名为“image”，类型为二进制文件。然后删除旧的Fugitive.h和Fugitive.m文件，通过菜单选项生成新文件。

如果你还是不清楚该怎样做，请查看第7章。

通往照相机之路

是使用UIImagePickerController。为什么呢？因为我们在此的真正使命是选择一个图像。iPhone是通过一个选取器来实现图像选择的，这个选取器可以让你从各种地方获得图像，比如照相机或者照片库。

UIImagePickerController类有许多内建的功能，而且它还是模式视图，所以一旦你用了它，许多事情就可以自动完成，而不必在应用中额外添加任何代码。



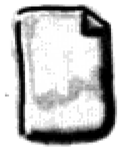
现在只剩下一些语法问题了……

【拼吾爱-最新编程资源的分享下载站】



即食代码

这是一些需要和图像选取器绑定在一起的代码。这些代码要放在CapturedPhotoViewController中，作为下一个练习的一部分。



CapturedPhotoViewController 提示*

```
- (void) viewWillAppear:(BOOL)animated {
      [super viewWillAppear:animated];
      self.fugitiveImage.image = [[[UIImage alloc]
        initWithData:fugitive.image] autorelease];
}
```

当视图出现时，如果数据库中有图像，就会将此图像分配给视图。

```
- (IBAction) takePictureButton: (id) sender {
      NSLog(@"Taking a picture.");


    UIImagePickerController* picker =
      [[UIImagePickerController alloc] init];
    picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary |
                        UIImagePickerControllerSourceTypeCamera;
    picker.delegate = self;
    picker.allowsEditing = YES;

    [self presentModalViewController:picker animated:YES];
}


- (void)imagePickerController:(UIImagePickerController *)picker
          didFinishPickingImage:(UIImage *)image
          editingInfo:(NSDictionary *)editingInfo
{
    self.fugitive.image = UIImagePNGRepresentation(image);

    [self dismissModalViewControllerAnimated:YES];
    [picker release];
}
```

这让用户可以编辑他们正在挑选的照片。

选取器会异步地显示出来。

选中一个图像后，就会调用这个方法。

移除选取器界面并释放选取器对象。

【拼吾爱-最新编程资源的分享下载站】

练习

是时候来一些图像啦！用我们给你的图像选取器代码，加上你自己的Objective-C能力，让图像功能运行起来。

1. **导入Fugitive类的头文件，为fugitive成员变量声明属性。**
   CapturedPhotoViewController需要了解自己处理的是哪一个逃犯。请在CapturedPhotoViewController类中添加一个Fugitive类型的成员变量和属性，并命名为“fugitive”。

2. **当图像被选中时存储图像，并更新UIImageView。**
   当选取器给出一个图像时，你需要在逃犯信息中设置图像的信息，然后还要确保视图被显示时UIImageView已被更新。你需要为UIImageView准备一个输出口；然后在Interface Builder将它们连接起来。

3. **添加takePictureButton动作方法中那些为UIImagePickerController准备的代码。**
   用我们给你的那些代码来完善对UIImagePickerController的支持。你需要说明CapturedPhotoViewController遵循UIImagePickerControllerDelegate和UINavigationControllerDelegate协议，才能让它做代理。

4. **添加“拍照按钮”。**
   用Interface Builder创建一个按钮，这个按钮需要覆盖整个UIImageView，然后再放到它后面去。别忘了将按钮与takePictureButton动作方法联系起来。

   创建按钮后，选中它并用菜单选择Layout→Send to Back选项。

5. **修改FugitiveDetailViewController的showInfoButtonPressed方法，在里面设置逃犯信息。**
   你需要在CapturedPhotoViewController创建后与被单击之前，将逃犯信息传递给它。

【拼吾爱-最新编程资源的分享下载站】

练习答案

下面就是将所有片段整合在一起，实现的按钮功能。

1 导入Fugitive类的头文件，为fugitive成员变量声明属性。

```
#import "Fugitive.h"

@interface CapturedPhotoViewController : UIViewController
<UINavigationControllerDelegate, UIImagePickerControllerDelegate> {
        UIImageView *fugitiveImage;
        Fugitive* fugitive;
}

@property (nonatomic, retain) IBOutlet UIImageView *fugitiveImage;
@property (nonatomic, retain) Fugitive *fugitive;

- (IBAction) doneButtonPressed: (id) sender;
- (IBAction) takePictureButton: (id) sender;

@end
```

我们需要一个输出口，这样就可以用选中的图像来更新UIImageView了。

CapturedPhotoViewController.h

2 当图像被选中时存储图像，并更新UIImageView。

```
#import "Fugitive.h"

@implementation CapturedPhotoViewController
@synthesize fugitiveImage, fugitive;

- (void) viewWillAppear:(BOOL)animated {
      [super viewWillAppear:animated];
      self.fugitiveImage.image = [[[UIImage alloc]
                  initWithData:fugitive.image] autorelease];
}
```

CapturedPhotoViewController.m

【拼吾爱-最新编程资源的分享下载站】

3 添加takePictureButton动作方法中那些为UIImagePickerController准备的代码。

```
- (IBAction) takePictureButton: (id) sender {
      NSLog(@"Taking a picture.");
      UIImagePickerController* picker =
          [[UIImagePickerController alloc] init];

    picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary |
                        UIImagePickerControllerSourceTypeCamera;

    picker.delegate = self;

    picker.allowsEditing = YES;


    [self presentModalViewController:picker animated:YES];
}
```

这里记录方法日志，好让我们能在调试器中看到它被调用了。

我们将代理设为自己，好在图像被选中（或者用户选择取消）时收到通知。

然后显示选取器，等待事情发生…

…而且不会释放选取器控制器，直到收到图像的回调为止。

```
- (void)imagePickerController:(UIImagePickerController *)picker
          didFinishPickingImage:(UIImage *)image
          editingInfo:(NSDictionary *)editingInfo
{
      self.fugitive.image = UIImagePNGRepresentation(image);


    [self dismissModalViewControllerAnimated:YES];
    [picker release];
}
```

Core Data要存储二进制数据，我们需要将原始图像信息从UIImage中提取出来。我们将其转换为PNG格式。

```
- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    [self dismissModalViewControllerAnimated:YES];
    [picker release];
}

- (void)dealloc {
      [fugitive release];
      [fugitiveImage release];
      [super dealloc];
}
@end
```

你要记住，得到图像之后，马上释放选取器控制器。

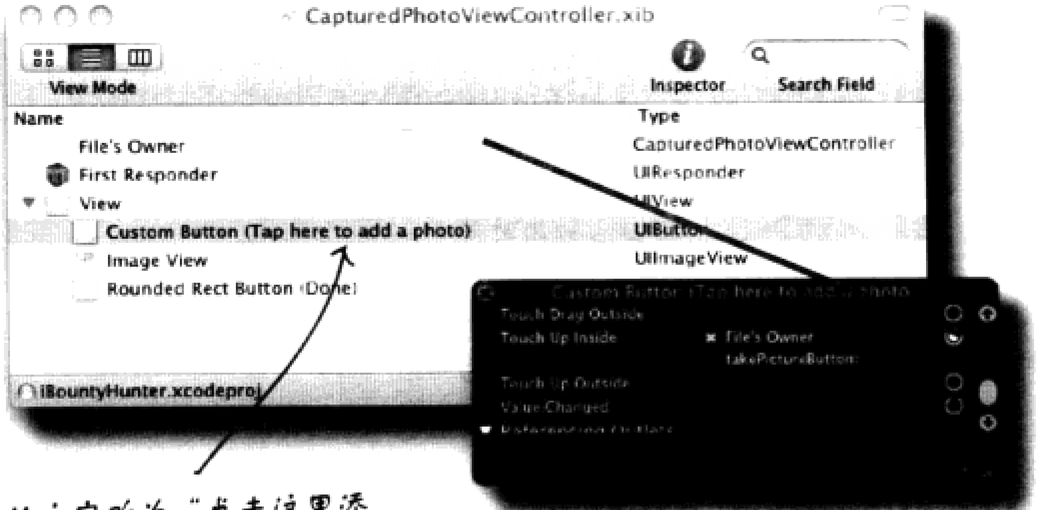
CapturedPhotoViewController.m

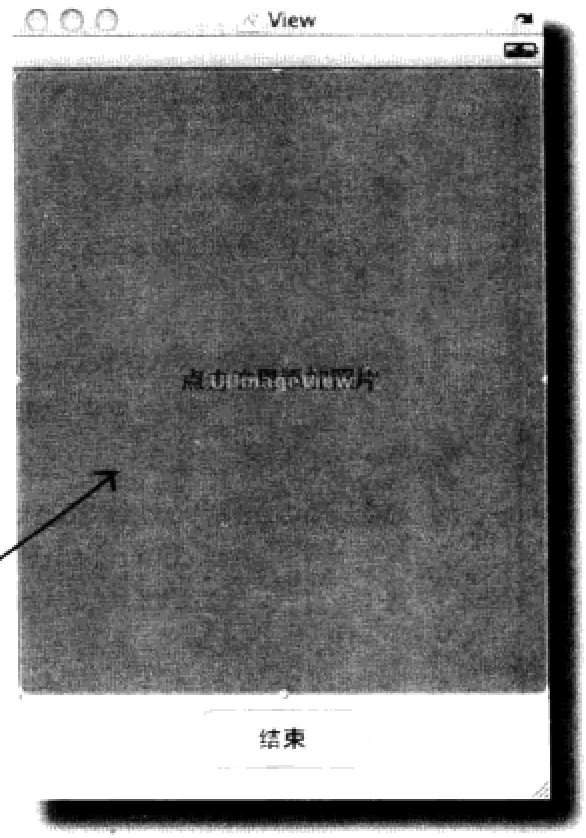
【拼吾爱-最新编程资源的分享下载站】



下面就是将所有片段整合在一起而实现的按钮功能……

4 添加“拍照按钮”。





将默认文字改为“点击这里添加照片”；因为它在图片后面，所以只有在没有图像的时候才会看到它。

这里看着有一点不太明显，不过按钮确实是在UIImageView后面的。

5 修改FugitiveDetailViewController的showInfoButtonPressed方法，在里面设置逃犯信息。

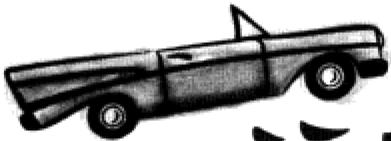
```
- (IBAction)showInfoButtonPressed: (id) sender {
    CapturedPhotoViewController *controller =
       [[CapturedPhotoViewController alloc] initWithNibName:
       @"CapturedPhotoViewController" bundle:nil];
    controller.fugitive = self.fugitive;
    controller.modalTransitionStyle =
      UIModalTransitionStyleFlipHorizontal;
    [self presentModalViewController:controller animated:YES];

    [controller release];
}
```

我们只要把逃犯信息设置到为CapturedPhotoViewController新添加的属性上就行了。

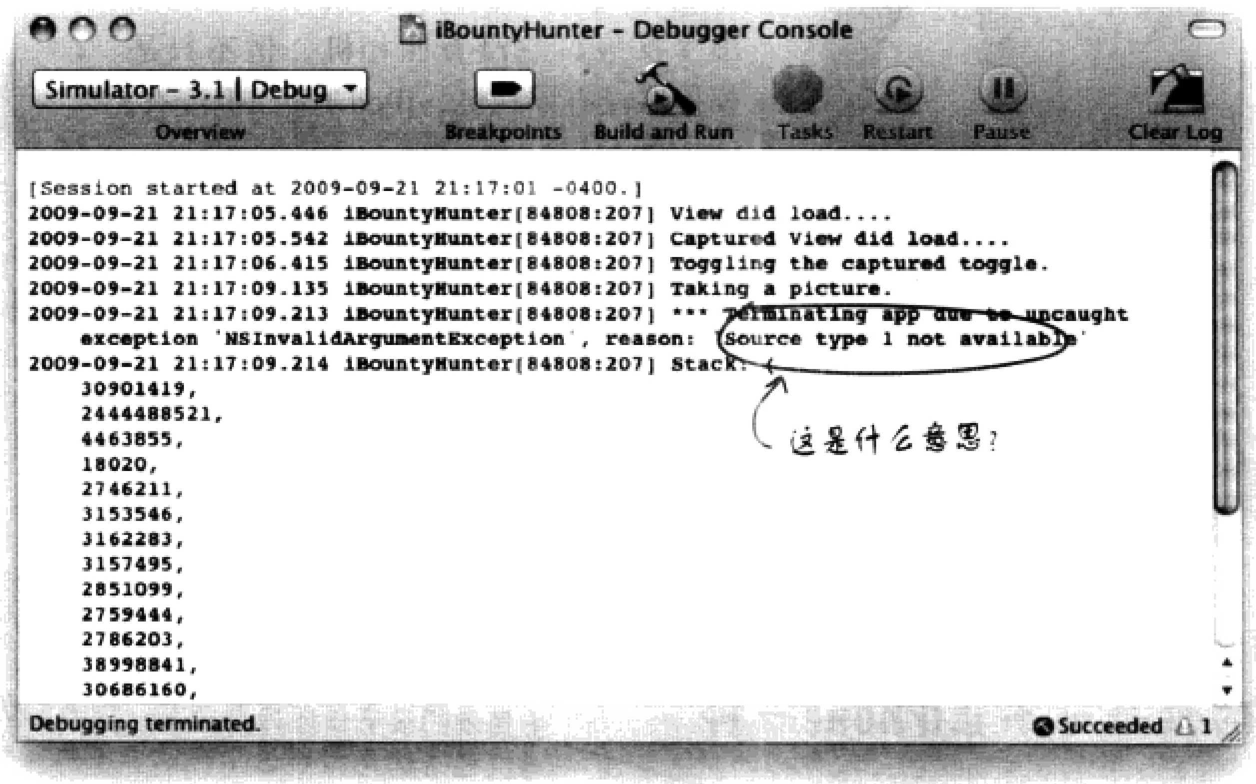
FugitiveDetailViewController.m

【拼吾爱-最新编程资源的分享下载站】



运行测试

构建并运行应用，看看实际运行中的新照片视图。



啊！崩溃啦！

脑力练习

哪里出错了？这里要跳出模拟器的范围考虑一下问题……

【拼吾爱-最新编程资源的分享下载站】



对了！而且iPod Touch也没有。

你要求使用照相机，而模拟器对此做出了反应，它没有照相机。但不只是模拟器没有照相机，iPod Touch也没有。

谁会在乎这些呢？苹果公司。

会使用应用的设备可不止iPhone一种

苹果公司要求发布的应用在所有能够运行该应用的设备上都能正常工作，这些设备目前包括iPod Touch和 iPhone。应用的审核过程中有一部分就是检查应用对iPod Touch的兼容性。

作者提示：

虽然我们没有任何内部消息，但我们仍然认为，随着时间推移这个列表会不断增长。

这些都意味着你要清楚地意识到何时应用会触及到iPhone与iPod Touch之间行为有差异的领域。

实际上到底有多少区别呢？

【拼吾爱-最新编程资源的分享下载站】

池中谜题

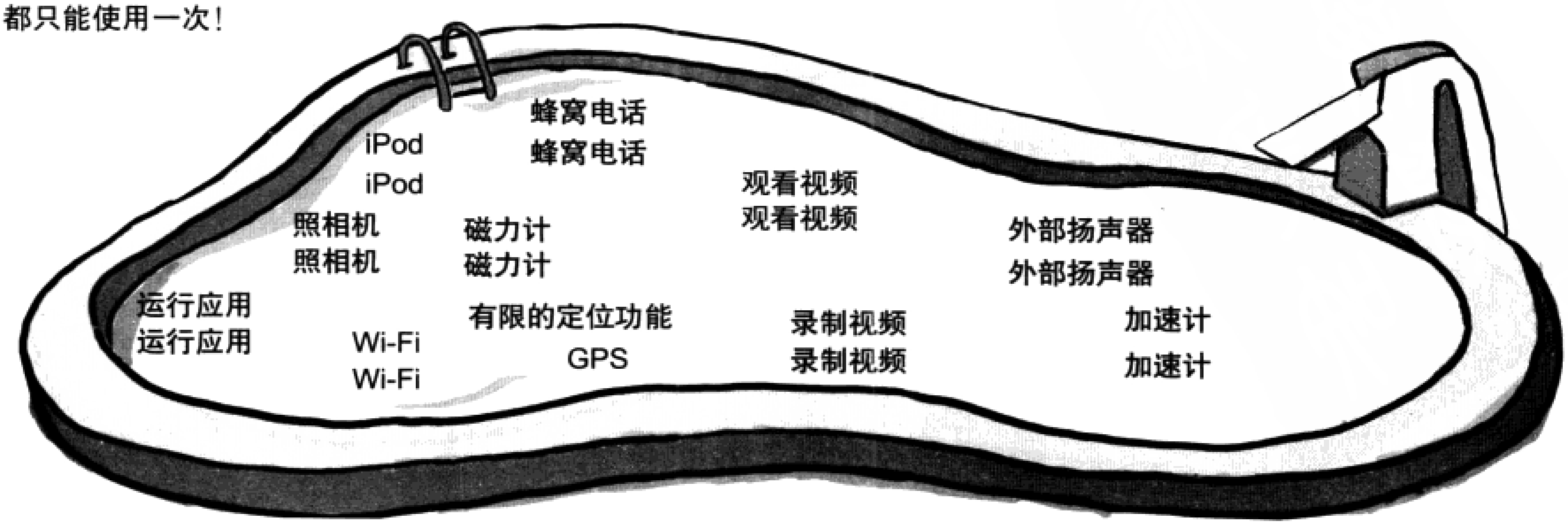


你的任务是将内容从池中取出来，并将它们放置到iPhone或iPod Touch的列表中。你不可以多次使用同一个内容，而且你不一定要使用完列出的所有内容。你的目标是弄出一个iPhone和iPod Touch功能的完整列表。

iPod Touch

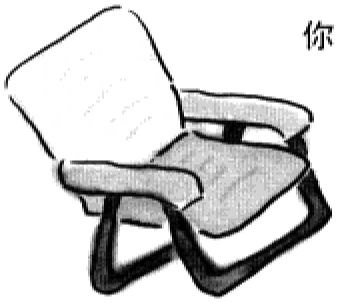
iPhone

注意：池中的每样东西都只能使用一次！



【拼吾爱-最新编程资源的分享下载站】

池中谜题答案



你的任务是将内容从池中取出来，并将它们放置到iPhone或iPod Touch的列表中。你不可以多次使用同一个内容，而且你不一定要使用完列出的所有内容。你的目标是弄出一个iPhone和iPod Touch功能的完整列表。



这个列表可能会发生变化。

苹果公司一直不断推出新设备并提供更多的功能。你需要不时检查一下，情况有没有变化。

iPod Touch	iPhone
iPod	iPod
运行应用	运行应用
观看视频	观看视频
有限的定位功能	GPS
加速计	加速计
Wi-Fi	Wi-Fi
	蜂窝电话
	照相机
	外部扬声器
	录制视频
	磁力计

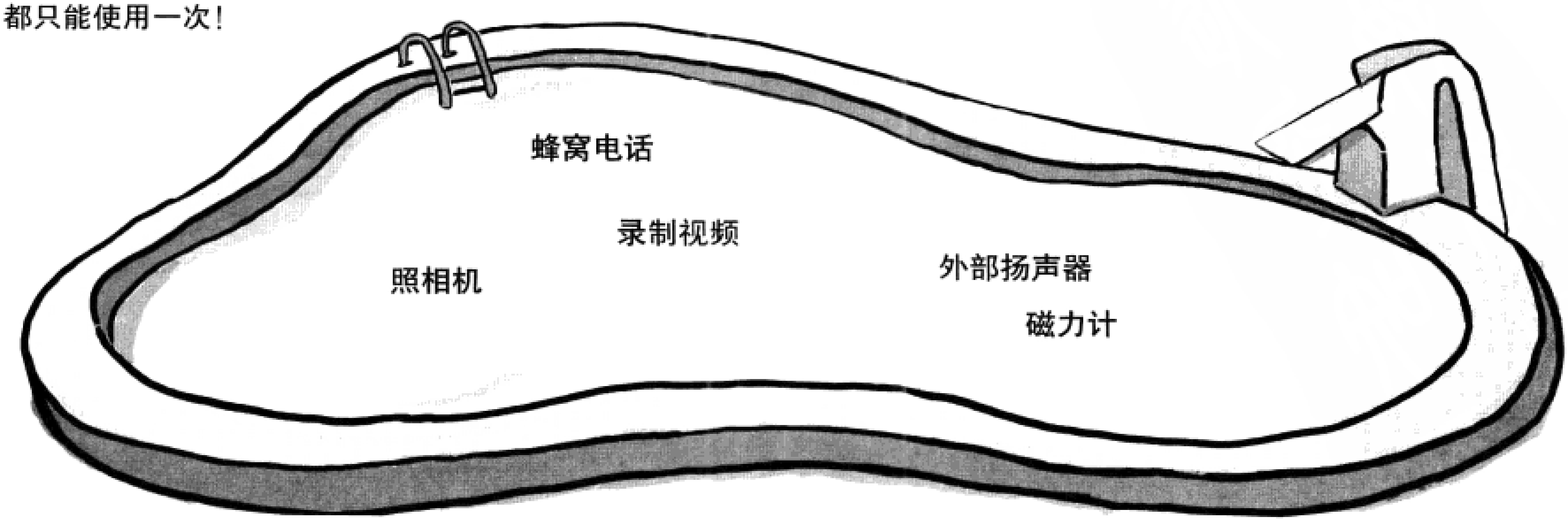
你可能偶尔也注意到了表中的一些东西了——但谁会去考虑扬声器的问题呢？

你可以通过Wi-Fi得到一些关于位置的信息。

这可能会是个问题。

只在3GS才有

注意：池中的每样东西都只能使用一次！



【拼吾爱-最新编程资源的分享下载站】

有一个方法可以进行检查

这些细微之处在各种设备之间都可能有差异，每次你要用到设备上的什么东西时，最好都检查一下，看看它是否存在于设备上。对于照相机，UIImagePickerController有一个方法可以进行检查。

```
[UIImagePickerController
  isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]
```

因为我们是从一个源中获得信息，所以需要检查需要的源是否存在。

对于我们来说，还有另一个选择：照片库。如果没有照相机，我们可以转而从那里获取图像。

脑力练习

那么当用户点击“添加照片”按钮时，到底发生了什么事情？你检查是否有照相机，然后怎么了？用户流程是什么样的？

【拼吾爱-最新编程资源的分享下载站】

用操作表单提示用户

操作表单会从页面的底部滑动上来，为用户提供一些选项，用户进行选择后继续运行。它类似于模式视图，因为用户必须对操作表单进行处置之后，才能继续进行其他动作。操作表单用起来很直接了当：它接受字符串作为按钮的内容，并内建有显示和消失的动画效果。我们的代码要用到操作表单的话，需要包含一些标准内容：

```
UIActionSheet *photoSourceSheet =
  [[UIActionSheet alloc] initWithTitle:
                    @"Select Fugitive Picture"
      delegate:self cancelButtonTitle:@"Cancel"
      destructiveButtonTitle:nil
      otherButtonTitles:@"Take New Photo", @"Choose
      Existing Photo", nil, nil];
```

首先分配操作表单并将标题传递给它。

所有操作表单都需要有一个取消按钮，这样你才能像对模式视图一样取消它。

这个按钮会被以红色高亮显示。我们不要这个。

操作表单常常有一个按钮，功能就是“我知道这样会删掉所有东西。照做吧”。这就是销毁按钮。

声明好其他两个按钮，就可以了。

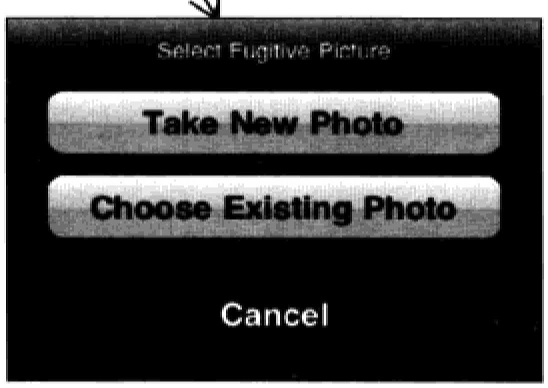
```
[photoSourceSheet showInView:self.view];
[photoSourceSheet release];
```

与UIImagePickerController不同，我们会立刻释放操作表单。

我们会用操作表单让用户挑选图像来源

我们知道选项有使用照相机、使用照片库和取消，所以我们需要为每个选项实现相应的行为。

按下这些按钮都会发生些什么事？



进入照相机应用，拍照然后回来，将你的新图像放到Fugitive中。只要你用了UIImagePickerControllerSourceTypeCamera，剩下的事情它全包了。

来到照片库，挑选一个图像然后回来，把图像放到Fugitive中。这里UIImagePickerControllerSourceTypePhotoLibrary会处理其余工作。

回到图像视图。

拿起笔来

实现这个操作表单！要考虑的东西很多，因为我们要稍微改变一点应用的流程。

❶ **修改takePictureButton动作方法，将操作表单包含进来。**

iBountyHunter需要检查照相机是否存在，如果存在，用户就要选择是使用照相机还是使用现有图片。如果不存在，应用应该直接使用照片库。

这里就要使用操作表单了。

❷ **实现操作表单的代理方法。**

下面这些就足以让你起步了。考虑一下case 1情况下需要的选项以及默认选项，还要确保释放了选取器并显示视图。也不要忘了在头文件中声明UIActionSheetDelegate。

```
- (void)actionSheet:(UIActionSheet *)actionSheet
    didDismissWithButtonIndex:(NSInteger)buttonIndex {
      UIImagePickerController* picker =
        [[UIImagePickerController alloc] init];
      picker.delegate = self;
      picker.allowsEditing = YES;

      switch (buttonIndex) {
              case 0:
                      NSLog(@"User wants to take a new picture.");
                      picker.sourceType =
                        UIImagePickerControllerSourceTypeCamera;
                      break;
```

❸ **增强代码的可读性！**

我们用3个#pragma把实现代码分开：takePictureButton代码、UIImagePickerController代码和操作表单的代理代码。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来答案

操作表单应该已经可以使用了，而你的应用的用户流程也更加合理了……

1 修改takePictureButton动作方法，将操作表单包含进来。

如果你想让操作表单在模拟器上也能使用，要把这个改成SourceTypePhotoLibrary。

```
- (IBAction) takePictureButton: (id) sender {
    NSLog(@"Taking a picture.");
    if ([UIImagePickerController
    isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {
        NSLog(
          @"This device has a camera, ask the user what they want to
do.");
        UIActionSheet *photoSourceSheet =
          [[UIActionSheet alloc] initWithTitle:@"Select Fugitive Picture"
                                  delegate:self cancelButtonTitle:@"Cancel"
                                  destructiveButtonTitle:nil
        otherButtonTitles:@"Take New Photo", @"Choose Existing Photo",
                            nil, nil];
        [photoSourceSheet showInView:self.view];
        [photoSourceSheet release];
    }
    else { // No camera, probably a touch
        NSLog(@"No camera available on the device.  Defaulting to library.");
        UIImagePickerController* picker = [[UIImagePickerController alloc] init];
        picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
        picker.delegate = self;
        picker.allowsEditing = YES;
        [self presentModalViewController:picker animated:YES];
    }
}
```

CapturedPhotoViewController.m

③ 实现操作表单的代理方法。

```
@interface CapturedPhotoViewController :
    UIViewController <UINavigationControllerDelegate,
    UIImagePickerControllerDelegate, UIActionSheetDelegate>
```

CapturedPhotoViewController.h

```
- (void)      actionSheet:(UIActionSheet
*)actionSheet didDismissWithButtonIndex:(NSInteger)buttonIndex {
      UIImagePickerController* picker =
        [[UIImagePickerController alloc] init];
      picker.delegate = self;
      picker.allowsEditing = YES;

      switch (buttonIndex) {
            case 0:
                  NSLog(@"User wants to take a new picture.");
                  picker.sourceType =
                    UIImagePickerControllerSourceTypeCamera;
                  break;
            case 1:
                  NSLog(@"User wants to use an existing picture.");
                  picker.sourceType =
                    UIImagePickerControllerSourceTypePhotoLibrary;
                  break;
            default:
                  // 用户选择了取消
                  [picker release];
                  return;
      }
      [self presentModalViewController:picker animated:YES];
}
```

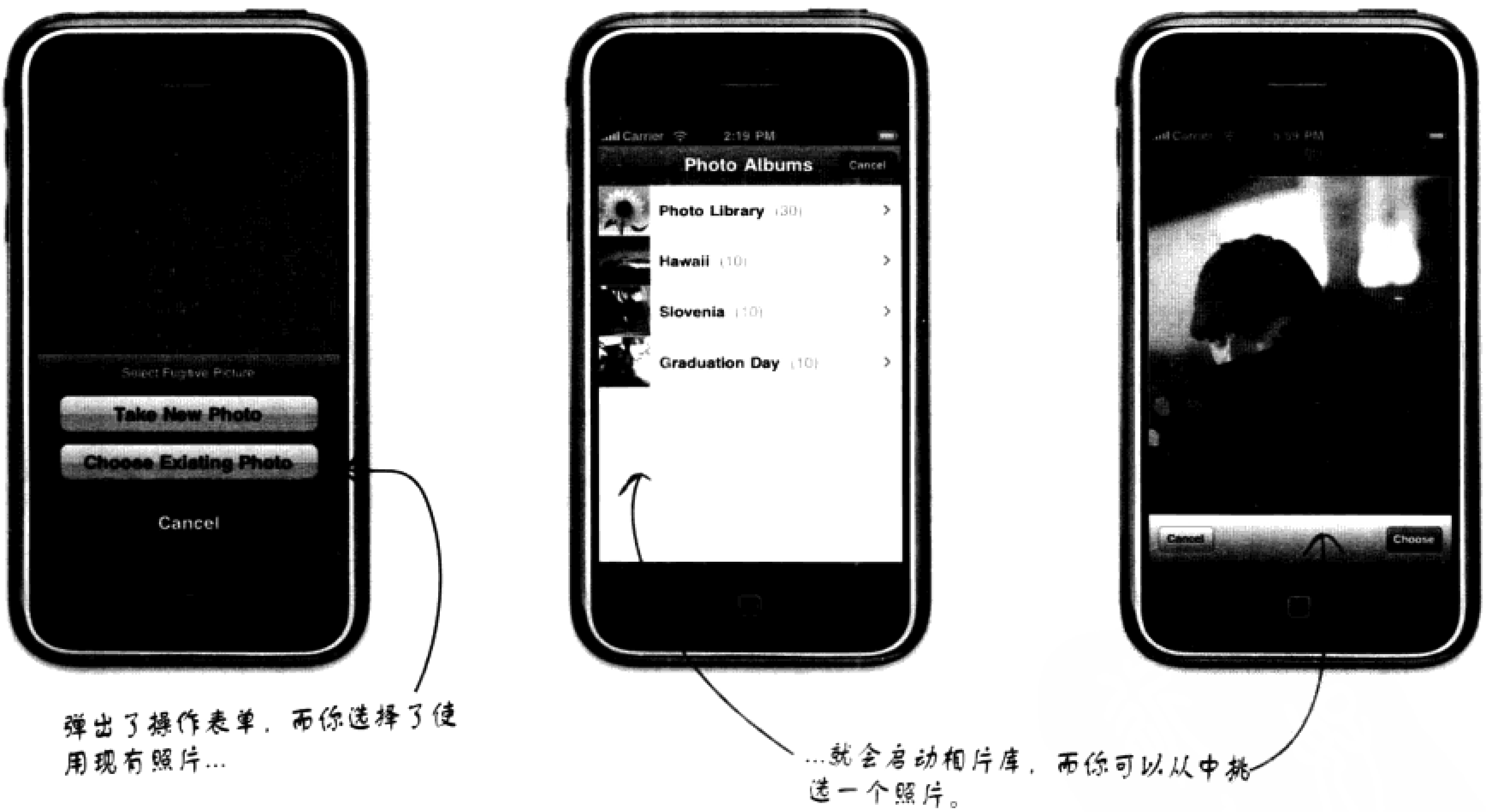
CapturedPhoto ViewController.m

能工作吗？

【拼吾爱-最新编程资源的分享下载站】

运行测试

启动iBountyHunter，选择一个逃犯后，进入拍照这个步骤。如果你在takePictureButton代码中使用的是SourceTypePhotoLibrary，一切都可以正常工作，你会见到这个操作表单。



极客点滴

现在应该去注册苹果公司的开发者计划了。如果你注册了，就可以将应用安装在真实的iPhone设备上进行测试了。请查看本书末尾的附录，里面的内容可以帮助你通过整个注册设备的过程，让它工作起来。

有问必答

问：现在iPhone 3GS不是支持视频了吗？我怎么才能用到它呢？

答： 这是另一种媒体类型，可以在使用UIImagePickerController时使用。默认情况下，会使用静止图像，也正是我们在iBountyHunter中想要的。

问：那照相机带的现实增强功能是怎么回事？我能做类似的功能吗？

答： 是。你可以把一个自定义覆盖视图交给UIImagePickerController，让它在调用照相机的时候使用。对于能在照相机视图中做的事是有限制的，不过如果愿意你还是可以用自己的信息来覆盖在上面的。

问：如果在使用UIImagePickerController的时候把allowEditing打开，有什么用？

答： 选取器控制器有内建支持，可以对图像进行裁剪和缩放。allowEditing标志控制着用户是否可以在将图像发送给代理之前，对其进行移动和修改大小。如果你打开了这个标志，而用户又修改了图像，你会在回调函数中得到编辑信息。

问：我们真的必须操心iPod Touch的问题吗？

答： 是的。当你将应用程序提交给苹果公司，要求将其加入iTunes App Store时，得指定你的应用程序能够用在哪些设备上。如果你声称支持某种设备，苹果公司就会在上面进行测试。他们还会进行一些无法访问网络情况下的测试，以确保你的应用程序可以正确处理这些情况。要考虑的周到一点。苹果公司会对你的应用程序进行各种场合下的测试。

问：有什么办法能在模拟器中测试照相机吗？

答： 没有。我们现在的所作所为已经尽可能地接近这一点了，可以用照片库来测试为照相机编写的代码。到目前为止你已经学到了很多东西，涉及的很多功能已经超出了模拟器能够处理的范围。GPS功能、加速计、扬声器功能，所有这些都无法在模拟器中测试，而要想真正地对它们进行测试，你就需要将应用安装到iPhone上。

问：那么能不能再说一次关于苹果公司开发者计划的事情？

答： 为了将应用安装到设备上，或者提交给App Store，你需要成为一个苹果公司的注册iPhone开发者。目前的费用是$99。即使你只是想为了个人目的将一个应用安装到设备上，也要注册才行。

请查看附录，那里有更多详细的指导，说明了如何将应用安装到你实际使用的电话上。

让我们给鲍勃看一看……

【拼吾爱-最新编程资源的分享下载站】

除了时间之外鲍勃还要知道地点

你已经给了鲍勃一种记录抓捕证据的方法，那就是用被捕者的照片，也可以很简单地记下抓捕发生的时间，但是地点怎么办呢？



鲍勃有管辖权的问题。

关于鲍勃在哪里可以抓罪犯是有规定的，所以他需要记录抓捕发生的地点。

为鲍勃记录这些内容的最简单的方法，就是记下抓捕发生的经纬度。

【拼吾爱-最新编程资源的分享下载站】

拿起笔来

新加两个字段会对应用造成什么样的影响？在下面的空白处，画出经度和纬度信息应该放在哪里，出现在哪个视图中。

在这里画草图

数据模型和数据自身又要发生哪些变化？

....

....

....

....

....

....

【拼吾爱-最新编程资源的分享下载站】

拿起笔来 答案

下面就是我们提出的新视图和数据变化：



数据模型和数据自身又要发生哪些变化？

数据库需要更新：我们会得到以度为单位的纬度值和经度值。为了保存在数据库中，需要将它们分解成Fugitive类的两个新属性：纬度和经度。

【拼吾爱-最新编程资源的分享下载站】

打造位置信息

进入应用之中，准备好接受抓捕的坐标：



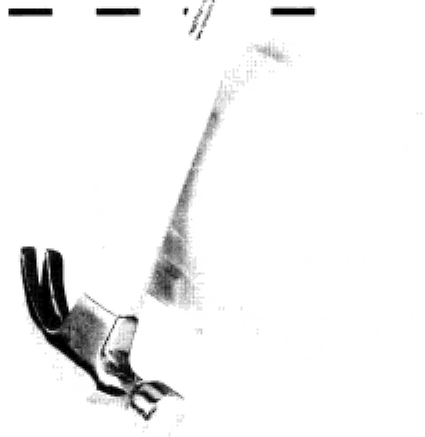
- [ ] 在视图中为位置标签和经度、纬度实现新字段。

- [ ] 再次迁移数据库，制造出新的具有经度、纬度成员的Fugitive类。

我们将其称为capturedlat和capturedlon，类型定为“Double”。

【拼吾爱-最新编程资源的分享下载站】

打造位置信息



进入应用之中，准备好接受抓捕的坐标：

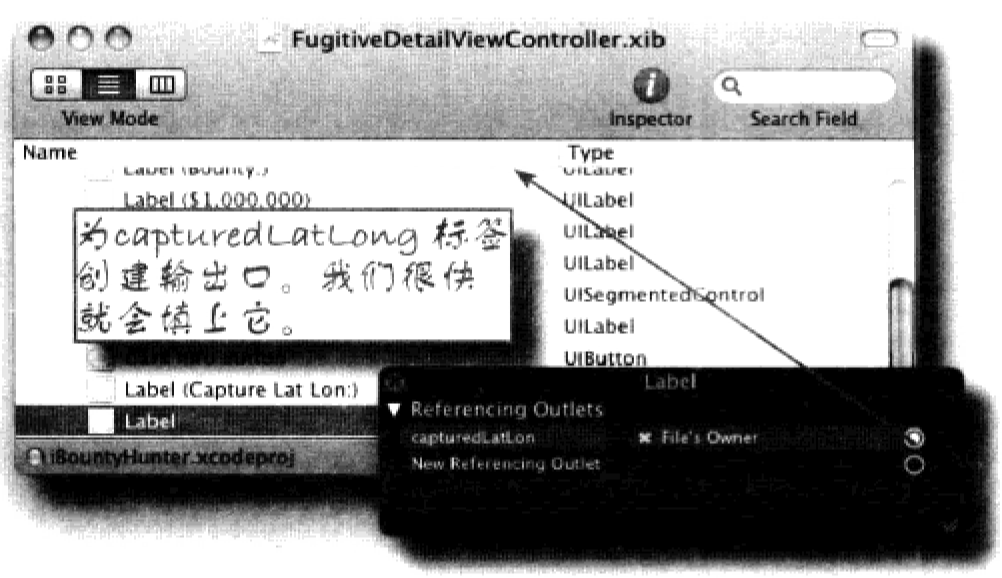
☑ 在视图中为位置标签和经度、纬度实现新字段。

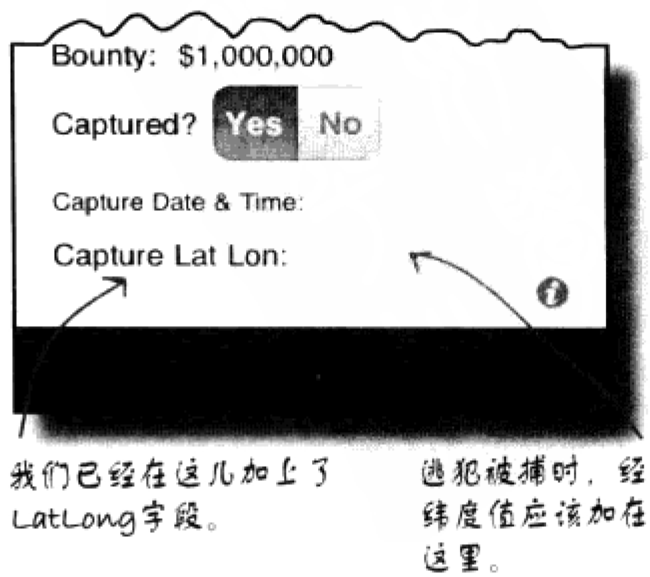
```
UILabel *capturedLatLon;
@property (nonatomic, retain)
  IBOutlet UILabel *capturedLatLon;
```

FugitiveDetailViewController.h

```
@synthesize capturedLatLon;
capturedLatLon.text = [NSString stringWithFormat:
  @"%.3f, %.3f", [fugitive.capturedLat doubleValue],
                 [fugitive.capturedLon doubleValue]];
[capturedLatLon release];
```

FugitiveDetailViewController.m

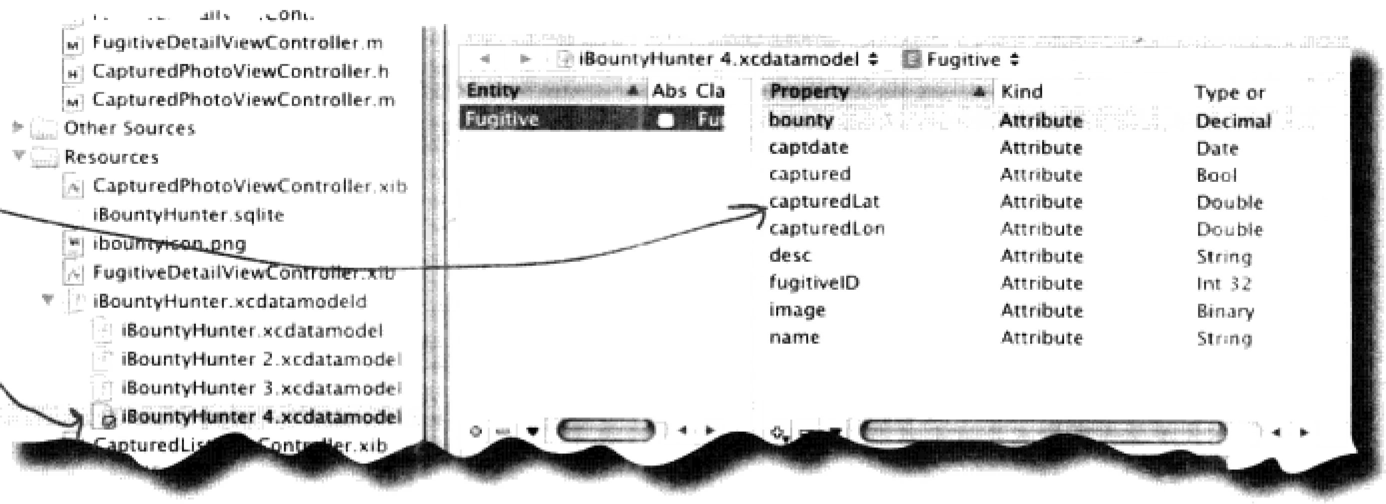




☑ 再次迁移数据库，制造出新的具有经度、纬度成员的Fugitive类。

新成员capturedLat和catpuredLon，类型都是“Double”。

我们已经达到了iBountyHunter 4.xcdatamodel。



好吧，我敢打赌你可以通过iPhone上的GPS获得这些信息，但你不是刚刚警告过我们iPod Touch没有这些功能吗？



确实如此，不过你可以另辟蹊径。

你可能还记得前面的池中谜题，里面说过iPod Touch可以有限地处理位置信息。iPhone（和iPod Touch）有不止一种方法可以得知你处于世界的哪个角落。

【拼吾爱-最新编程资源的分享下载站】

Core Location能以多种方式找到你

大多数人第一印象就会想到GPS，但是第一代iPhone没有GPS，而且iPod Touch也没有。这并不意味着你就没辙了。实际上iPhone有三种方式可以确定你的位置：GPS、蜂窝基站三角定位和Wi-Fi位置服务。

GPS是最精确的，其次是基站定位，最后是Wi-Fi。各款iPhone分别可以使用这些方法中的两种或三种，而iPod Touch只能用Wi-Fi，但这也比什么都没有强。Core Location会根据设备的状态和你需要的精度，决定使用哪种方法。也就是说，不用为它们检查来源是否存在等问题，iPhone OS自己会处理。

分配CLLocationManager

你需要传入精度。对于鲍勃来说10米就足够了。

```
self.locationManager = [[CLLocationManager alloc] init];
self.locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters;
self.locationManager.delegate = self;
[self.locationManager startUpdatingLocation];
```

一旦locationManager得到了位置，就会开始向代理发送信息，让你使用。

Core Location依靠LocationManager

要使用Core Location，你只要创建一个定位管理器，并请求它开始发送信息更新就可以了。根据你的设备的能力所及，它可以提供位置、高度和方向信息。要让它给你发送这些信息，你需要为其提供一个代理以及你要求的精度。CLLocationManager会在位置可用或者出错时通知你。你也要保证在无法从定位管理器获得位置时，能够正确进行处理。即使设备支持定位功能，在你收集位置信息之前系统也会提示用户是否允许记录他们的位置，而用户可以禁止这样做（故意的，或者偶然的）。

脑力练习

我们该在应用中哪个位置实现这些代码？



是的，还有一个新框架。

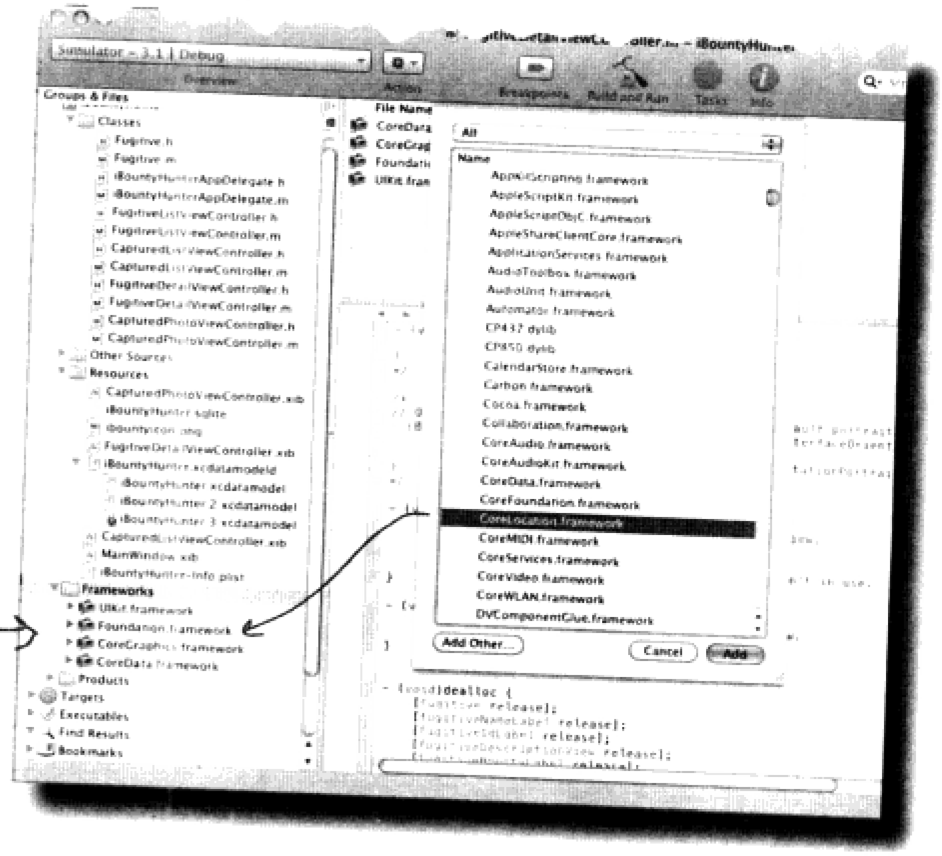
为了让应用的尺寸能更小一些，苹果公司将各种功能分解成多个库。当你开始加入新功能时，比如Core Location，就需要添加框架。由于Core Location框架默认没有被包括在应用中，我们需要把它加进来。

【拼吾爱-最新编程资源的分享下载站】

添加新框架

目前我们还很娇惯自己，只用过默认框架，要不然就是模板引入的框架。现在我们要拓展自身能力，将Core Location框架添加到应用中。

选中框架文件夹，然后单击右键，在弹出菜单中依次选择**Add → Existing Frameworks**...选项。然后选择“Core Location”并点击**Add**。



新框架会在这里列出。

然后更新头文件

我们还需要声明自己遵循CLLocationManagerDelegate协议，并加入属性。

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
#import "Fugitive.h"

@interface FugitiveDetailViewController : UIViewController
<CLLocationManagerDelegate> {

    CLLocationManager *locationManager;

}

@property (nonatomic, retain) CLLocationManager *locationManager;
```

包含新的CoreLocation框架。

我们通过代理完成工作，所以要有这个。

声明CLLocationManager，这样我们就可以在FugitiveDetailViewController.m中合成并使用它。

FugitiveDetailViewController.h



人人都是开发者

你的任务是作为开发者，想办法找出在用户流程的哪个位置来实现对Core Location的应用。前提是：鲍勃需要用位置和日期时间来标记一次抓捕行动。

1. 细节视图中的哪个方法会用来启动Core Location?

2. 当位置被返回给视图控制器时，会发生什么?

3. 如果Core Location无法得到信息或者用户禁用了位置功能，会发生什么?

4. 该在何时关闭Core Location功能?

5. 其他设备怎么办?



Core Location很耗电。

在你的应用中频繁地调用寻找位置功能，会很快耗尽电池，因为这样会打开GPS/移动电话/Wi-Fi接收器。这样会激怒用户，在iTunes上带来负面评价。尽量克制使用!

【拼吾爱-最新编程资源的分享下载站】

人人都是开发者



你的任务是作为开发者，想办法找出在用户流程的哪个位置来实现对Core Location的应用。前提是：鲍勃需要用位置和日期时间来标记一次抓捕行动。

❶ 细节视图中的哪个方法会用来启动Core Location?

将初始化Core Location的代码放在细节视图的viewWillAppear中。

❷ 当位置被返回给视图控制器时，会发生什么？

我们会知道定位管理器可以得到当前位置。如果用户将逃犯标为被捕，我们需要从定位管理器得到当前位置并更新逃犯信息。

```
- (IBAction) capturedToggleChanged: (id) sender {
      NSLog(@"Toggling the captured toggle.");
      if (capturedToggle.selectedSegmentIndex == 0) {
            NSLog(@"Dude got captured.");
            NSDate *now = [NSDate date];
            fugitive.captdate = now;
            fugitive.captured = [NSNumber numberWithBool:YES];
            CLLocation *curPos = self.locationManager.location;
            fugitive.capturedLat =
              [NSNumber numberWithDouble:curPos.coordinate.latitude];
            fugitive.capturedLon =
              [NSNumber numberWithDouble:curPos.coordinate.longitude];
      }
      else {
            fugitive.captdate = nil;
            fugitive.captured = [NSNumber numberWithBool:NO];
            fugitive.capturedLat = nil;
            fugitive.capturedLon = nil;
      }

      capturedDateLabel.text = [fugitive.captdate description];
      capturedLatLon.text = [NSString stringWithFormat:@"%.3f, %.3f",
        [fugitive.capturedLat doubleValue],
        [fugitive.capturedLon doubleValue]];
}
```

我们不需要持续更新位置，所以在用户切换被捕状态控件时，会请求定位管理器发送其最后的位置。

记住，由于Core Data对所有内容都使用对象，所以我们实际在逃犯信息中存储的是NSNumbers。

FugitiveDetailViewController.m

3 如果Core Location无法得到信息或者用户禁用了位置功能，会发生什么？

由于鲍勃在将一个逃犯标成被捕时需要位置信息，如果无法得到这些信息，我们需要将被捕状态控件禁用。

```
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation {
    NSLog(@"Core location claims to have a position.");
    capturedToggle.enabled = YES;
}

- (void)locationManager:(CLLocationManager *)manager
      didFailWithError:(NSError *)error {
    NSLog(@"Core location says no-go on the position info.");
    capturedToggle.enabled = NO;
}
```

由于分段控件被禁用时外表上没有对应的变化，你可能得考虑用一个UIAlertView警告用户，让他们知道无法将任何人标记为被捕。

FugitiveDetailViewController.m

4 该在何时关闭Core Location功能？

我们会在离开细节视图的时候关闭它。

```
-(void) viewWillDisappear:(BOOL)animated {
      [super viewWillDisappear:animated];

      NSLog(@"Shutting down core location...");
      [self.locationManager stopUpdatingLocation];
      self.locationManager = nil;
}
```

FugitiveDetailViewController.m

5 其他设备怎么办？

我们要做的就是告诉Core Location需要的精度，剩下的框架会处理。iPod Touch会尽量获得精确的数据，我们一样能用。

将这些代码全部实现，然后运行一下试试……

按图索骥！

【拼吾爱-最新编程资源的分享下载站】

有问必答

问：我们在viewWillAppear和viewWillDisappear中启动和停止Core Location。这正常吗？

答：很正常，你应该按照需要启动和停止Core Location。它在运行时，消耗的电力相当可观，所以不需要时最好将其关闭。这样做可能会有一点小麻烦，因为Core Location需要一些时间才能得到初始的位置信息。为了让用户体验更加流畅，我们在视图一显示出来后就打开它，让它在用户需要位置之前有些准备时间。

问：有没有办法可以加速获得初始位置呢？

答：Core Location会试着缓存之前的位置信息，以便能尽快地提供给你一些东西。所以，如果你真的非常关心定位精确问题，应该检查随着位置信息一起发送过来的时间戳，确认这个位置是最近得到的，符合你的需要。

问：定位精度会影响启动时间或者电池用量这些方面吗？

答：当然会。你请求的位置越精确，Core Location用的电力就越多，而且也就越有可能需要更长的时间来定位。信息的可信度越低，就越容易快速获得。你的应用程序需要什么精确度都可以用，但是你要清楚，高精度信息意味着什么。

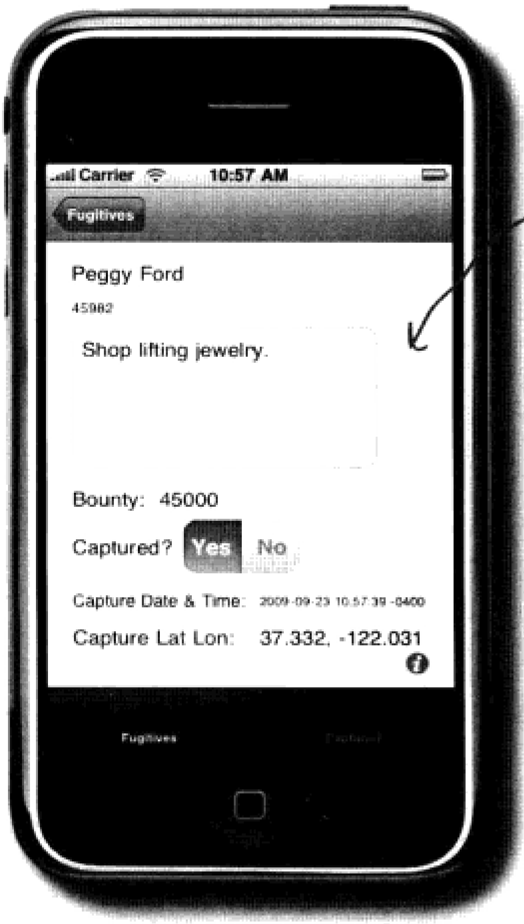
问：有没有办法等待Core Location返回一个位置，而不必让它像现在这样回调代理。

答：没有。Core Location与iPhone OS中许多其他框架一样，会在数据可用时异步地进行回调。网络访问通常也是以这种方式进行的。你需要确保用户知道应用程序在做什么，以及他们当前可以和不可以做什么。例如，在位置信息不可用时，我们会禁用被捕按钮。其他方式包括显示等待指示器（就像一个旋转的齿轮），或者用一个禁用的指示器来显示位置状态，比如图标、按钮和标签等。

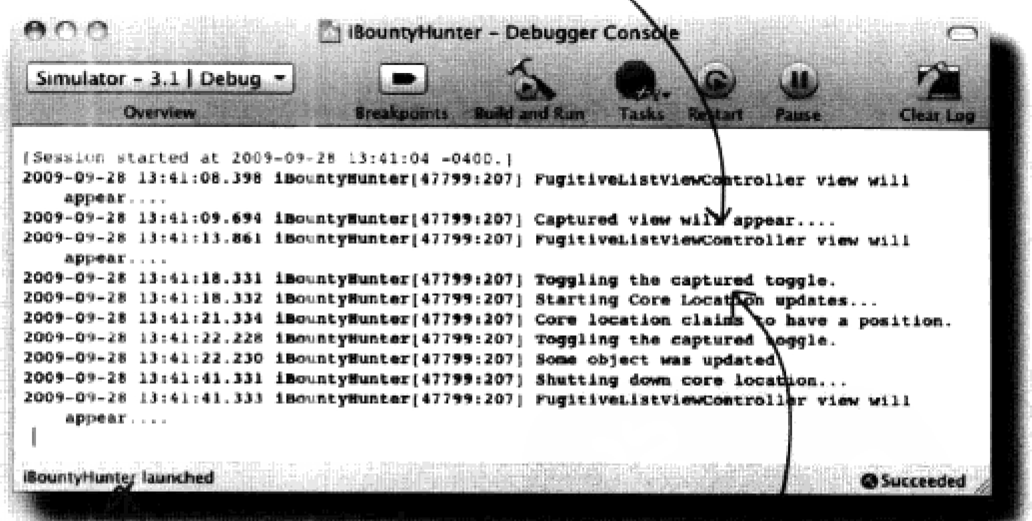
【拼吾爱-最新编程资源的分享下载站】

运行测试

实现Core Location真的没什么难度，但是要让它在用户流程中工作还需要再多做一些工作。现在一切都完成了，你应该可以运行了……



要操作这里，鲍勃得进入细节视图，这样就会启动Core Location管理器。



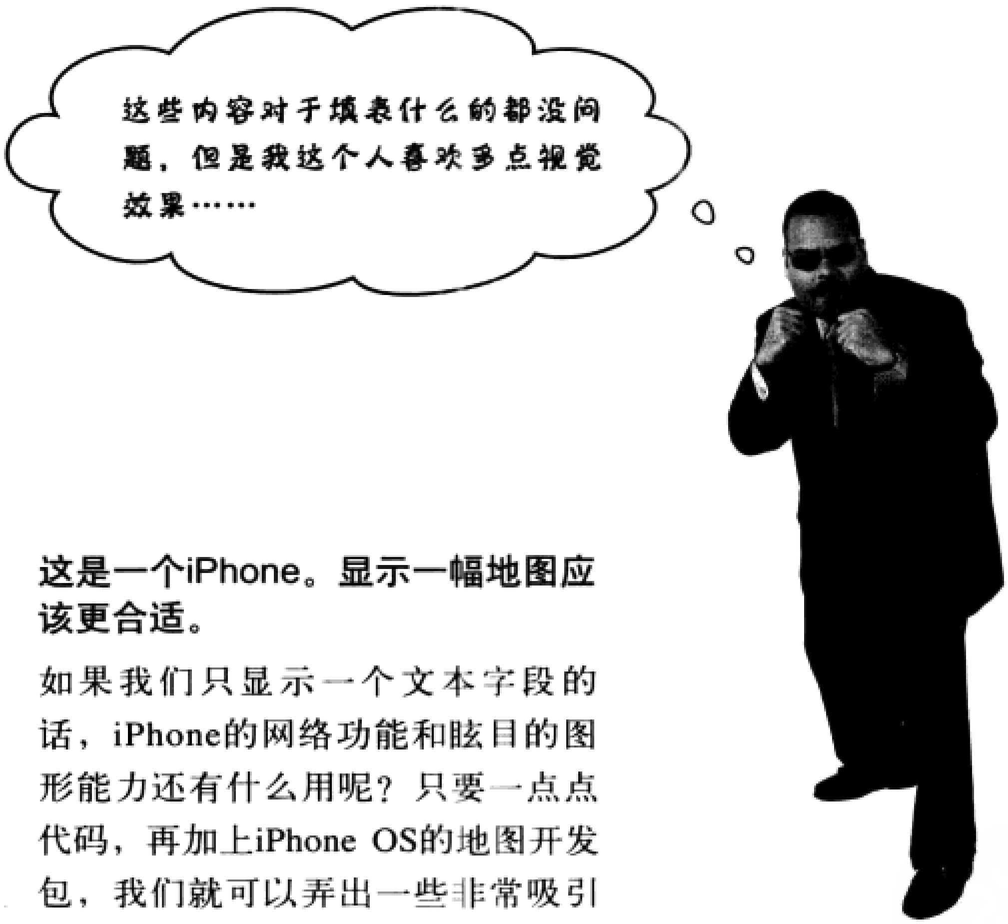
一旦有一个位置被返回了，被捕状态按钮就会启用，字段也会被填进数据。

如果你在viewWillAppear中加上了capturedToggle.enabled = NO;，那么在Core Location开始返回位置更新数据之前，用户无法使用这个控件。

工作正常！鲍勃应该做好准备……

【拼吾爱-最新编程资源的分享下载站】

仅仅有经纬度对于鲍勃是不够的



这是一个iPhone。显示一幅地图应该更合适。

如果我们只显示一个文本字段的话，iPhone的网络功能和眩目的图形能力还有什么用呢？只要一点点代码，再加上iPhone OS的地图开发包，我们就可以弄出一些非常吸引人的东西。

地图开发包是iPhone 3.0中新加入的

在最近一次iPhone主版本更新中，苹果公司开放了iPhone上面使用的地图API。地图数据来自Google Maps，其中包括了卫星影像。

对于地图你可以进行许多定制操作，比如显示区域的宽度、从何种视图开始显示以及大头针和注释功能。

使用地图开发包在逻辑上与Core Location非常类似：你需要一个新框架，还要在头文件中`#import <MapKit/MapKit.h>`。

MKMapView是从Google Maps获取地图信息的控件。你可以配置它显示一般的道路地图、卫星影像或者混合模式，就像这里看到的一样。



根据要在地图上显示的信息，你可以创建自己的注释视图，可以显示需要的任何内容，比如图片、格式化文本等。

地图开发包内置支持在特定位置插上大头针，称为注释。

留神！

地图开发包需要网络连接。

由于地图开发包从Google下载影像信息，所以你需要具有网络连接才能更好地使用它。这对于模拟器不受影响（假设你的Mac是联网的），但是对于iPod Touch甚至是iPhone来说，根据你所处的位置不同，有可能是个问题。地图开发包可以很好地对此进行处理，但是你还是得清楚这一点。

我们怎样才能用上它呢？

【拼吾爱-最新编程资源的分享下载站】

对地图进行一点自定义设置

与Core Location类似，在iBountyHunter中获得基本的地图开发包功能并不需要做很多工作。我们会更新CapturedPhotoViewController的viewWillAppear，以混合模式（卫星加上道路信息）地图来显示抓捕地点。

```
- (void) viewWillAppear:(BOOL)animated {
      [super viewWillAppear:animated];
      self.fugitiveImage.image =
        [[[UIImage alloc] initWithData:fugitive.image] autorelease];

      if ([fugitive.captured boolValue] == YES) {
            CLLocationCoordinate2D mapCenter;
            mapCenter.latitude = [fugitive.capturedLat doubleValue];
            mapCenter.longitude = [fugitive.capturedLon doubleValue];

            MKCoordinateSpan mapSpan;
            mapSpan.latitudeDelta = 0.005;
            mapSpan.longitudeDelta = 0.005;

            MKCoordinateRegion mapRegion;
            mapRegion.center = mapCenter;
            mapRegion.span = mapSpan;

            self.fugitiveMapView.region = mapRegion;
            self.fugitiveMapView.mapType = MKMapTypeHybrid;

      }
}
```

这里我们会传入逃犯被捕地点的经纬度值。

这些值让我们可以配置显示出来的默认地图的尺寸。

地图尺寸的单位是度。我们想让地图以相当大的比例进行放大。

我们把这些信息放在一起，来对地图进行初始化。

地图有几种类型：混合型同时具有卫星和道路信息。

这里我们将地图设置到视图中。

CapturedPhotoViewController.m

【拼吾爱-最新编程资源的分享下载站】

有问必答

问： Core Location和地图开发包之间有什么区别？

答： 地图开发包是显示地图用的，是对位置敏感的用户界面。Core Location则是用来获取你所在的位置信息的。你可以在Interface Builder里将一个地图拖拽到视图中；只要传入一些值，它就可以工作了。

另一方面，Core Location则将值返回给代理，你需要自己决定如何处理它们。例如，我们会从Core Location得到这些信息，并将其传给地图开发包，为我们显示抓捕地点的地图。

问： 这些框架都是从哪儿来的？如果我要用的框架没列出来怎么办？

答： 这些框架是SDK的一部分。框架的实际路径会随版本和平台而变化。例如，我们用的地图开发包框架的位置是：/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS3.1.sdk/System/Library/Frameworks/MapKit.framework。通常来说，你应该可以用“Add Existing Framework”选项来添加框架，而不必关心特定的位置，不过如果框架没有列出来，或者你要添加一个自定义框架，可以为Xcode指出实际路径。

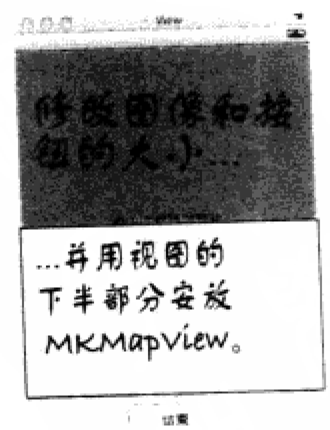


练习

实现地图功能，显示逃犯被捕地点所在区域。

1. **添加地图开发包框架和#import语句。**
添加框架的方法与添加Core Location一样。完成以后，确认你在细节视图中加入了#import语句，来包含地图开发包的头文件。

2. **配置照片视图来显示地图。**
我们并不添加一个完整的新视图，而是将地图直接加入到`CapturedPhotoView`中，和图像放在一起。改变图像和按钮的大小，然后将一个MKMapView拖到视图的下半部分。



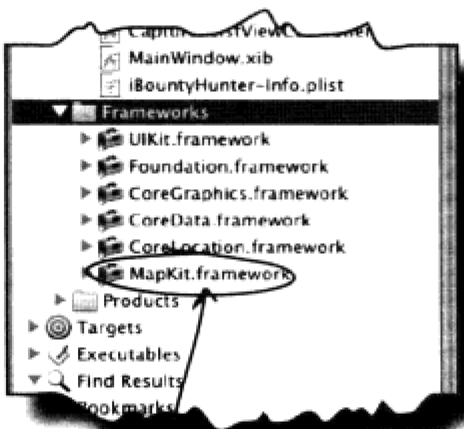
3. **为MKMapView添加输出口和代码。**
现在支持地图功能的内容全都已经就位了，继续把输出口和我们给你的地图代码添加进去，就能让地图功能工作起来。你要确保在Interface Builder中正确连接了输出口。

【拼吾爱-最新编程资源的分享下载站】

练习答案

实现地图功能，显示逃犯被捕地点所在区域。

1 添加地图开发包框架和#import语句。



这是地图开发包框架...

```
#import <MapKit/MapKit.h>

@class Fugitive;

@interface CapturedPhotoViewController :
  UIViewController <UINavigationControllerDelegate,
  UIImagePickerControllerDelegate, UIActionSheetDelegate> {
      MKMapView *fugitiveMapView;
}

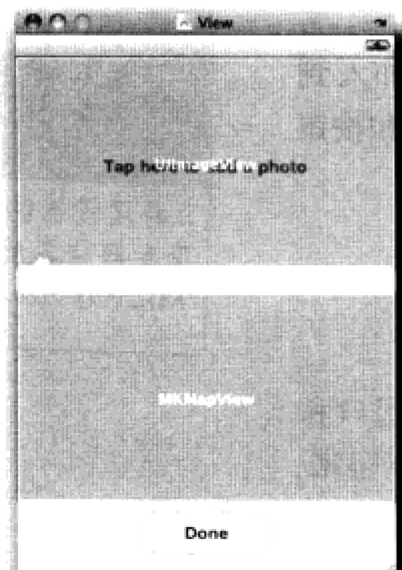
@property (nonatomic, retain)
    IBOutlet MKMapView *fugitiveMapView;
```

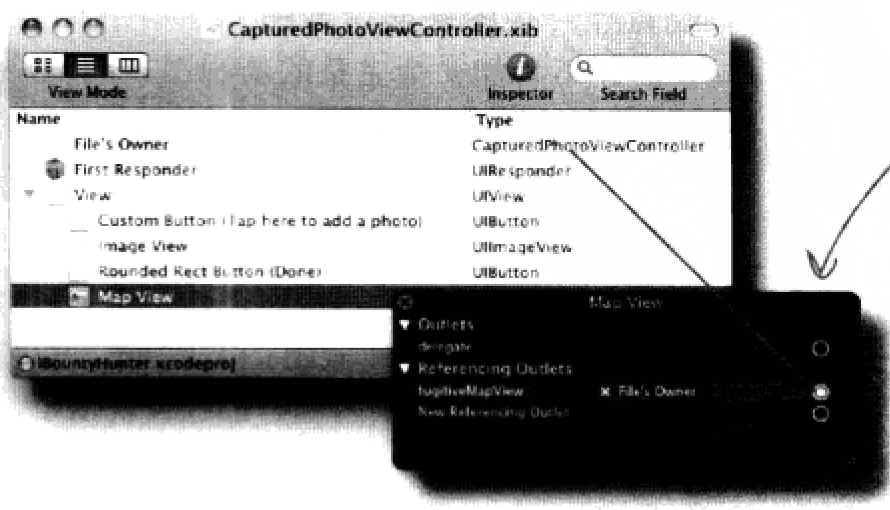
3 为MKMapView添加输出口和代码。



CapturedPhoto ViewController.h

2 配置照片视图来显示地图。





【拼吾爱-最新编程资源的分享下载站】

3 为MKMapView添加输出口和代码。

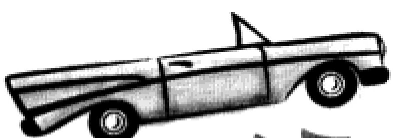
```
@synthesize fugitiveImage, fugitive, fugitiveMapView;
```

加入第474页的所有代码，对地图进行定制。

```
- (void)dealloc {
        [fugitive release];
        [fugitiveImage release];
        [fugitiveMapView release];
        [super dealloc];
}
@end
```



CapturedPhotoViewController.m



运行测试

继续构建并运行应用。你要确保已经将一个逃犯标为被捕状态，而经纬度字段也填好了，然后将视图翻转，看看地图。试着在地图上进行放大，在实际设备上，你可以用“捏合”的手势。在模拟器中，可以按住选项键不放，然后单击。

【拼吾爱-最新编程资源的分享下载站】

运行测试

试着在地图上进行放大，在实际设备上，你可以用“捏合”的手势。在模拟器中，可以按住选项键不放，然后单击。



由于你在用模拟器，不管你在哪儿，位置都将是在加州的Cupertino。

太棒了！现在我们只需一个大头针就可以显示抓捕是在哪里发生的。

注释功能还需要一点~~工作~~雕琢

注释是一个小小的标记，可以标出你感兴趣的点，样子是一个大头针。要使用注释功能，就要遵循地图开发包的注释协议。地图开发包使用注释协议，这样你就可以拿现成的类直接提供给地图开发包来用。这样做的不利之处是我们需要在Fugitive类中添加代码。

Fugitive.h

```
#import <MapKit/MapKit.h>
@interface Fugitive : NSManagedObject <MKAnnotation>
{
}
#pragma mark -
#pragma mark MapKit Annotation Protocol

@property (nonatomic, readonly)
CLLocationCoordinate2D coordinate;
- (NSString *) title;
- (NSString *) subtitle;
@end
```

如果是用于一个可能要进行多次数据迁移的应用，应该实现一个单独的类来遵循这个协议，里面放一个对Fugitive类的引用（合成），而不是直接把代码加进Fugitive类中。

```
(CLLocationCoordinate2D) coordinate {
    CLLocationCoordinate2D captureCoord;
    captureCoord.latitude =
      [self.capturedLat doubleValue];
    captureCoord.longitude =
      [self.capturedLon doubleValue];

    return captureCoord;
}
- (NSString *) title {
    return self.name;
}
- (NSString *) subtitle {
    return self.desc;
}
@end
```

Fugitive.m

按图索骥！

协议要求我们有一个coordinate属性、title和subtitle方法。我们没有合成这个coordinate属性，而是自己实现了它，将逃犯的位置和姓名等作为属性返回。

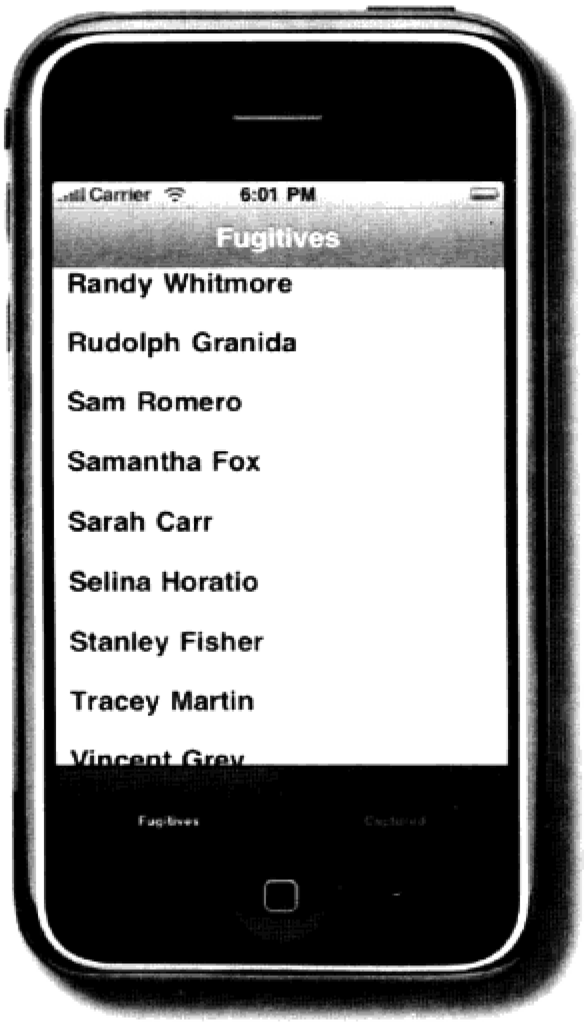
将这行代码添加在CapturedPhotoViewController.m中viewWillAppear方法的末尾。

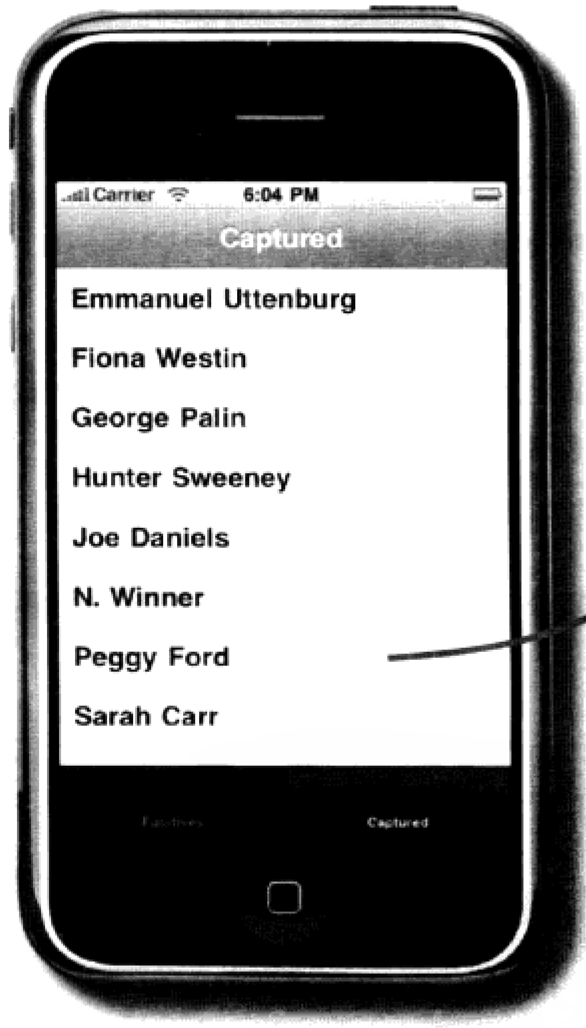
```
[self.fugitiveMapView addAnnotation:fugitive];
```

【拼吾爱-最新编程资源的分享下载站】

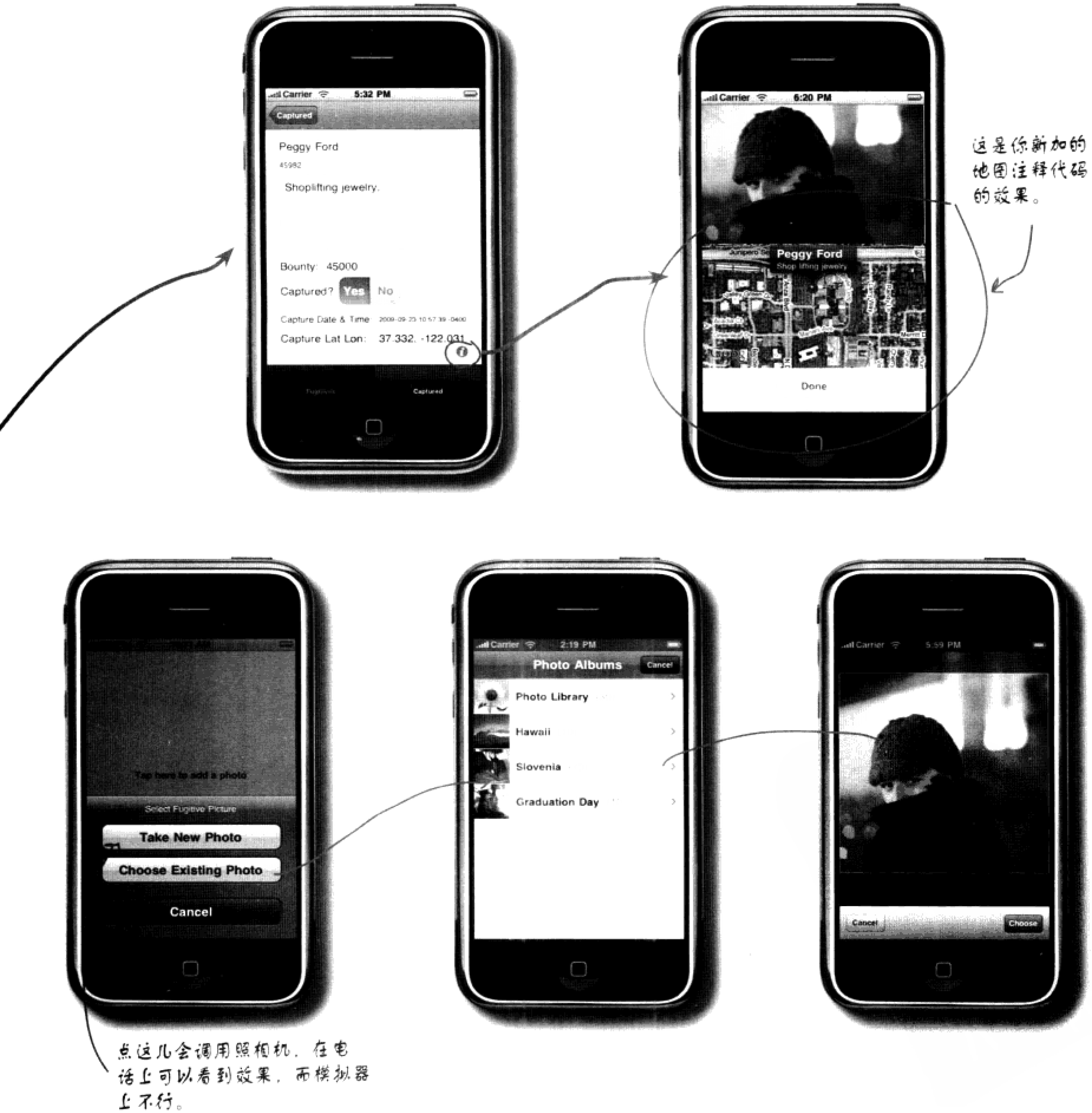
iBountyHunter 运行测试

完成了！现在所有功能都工作正常了。因为你一直都在使用这些代码，所以可能没有留意到，这个应用很大，也很棒！





【拼吾爱-最新编程资源的分享下载站】



【拼吾爱-最新编程资源的分享下载站】

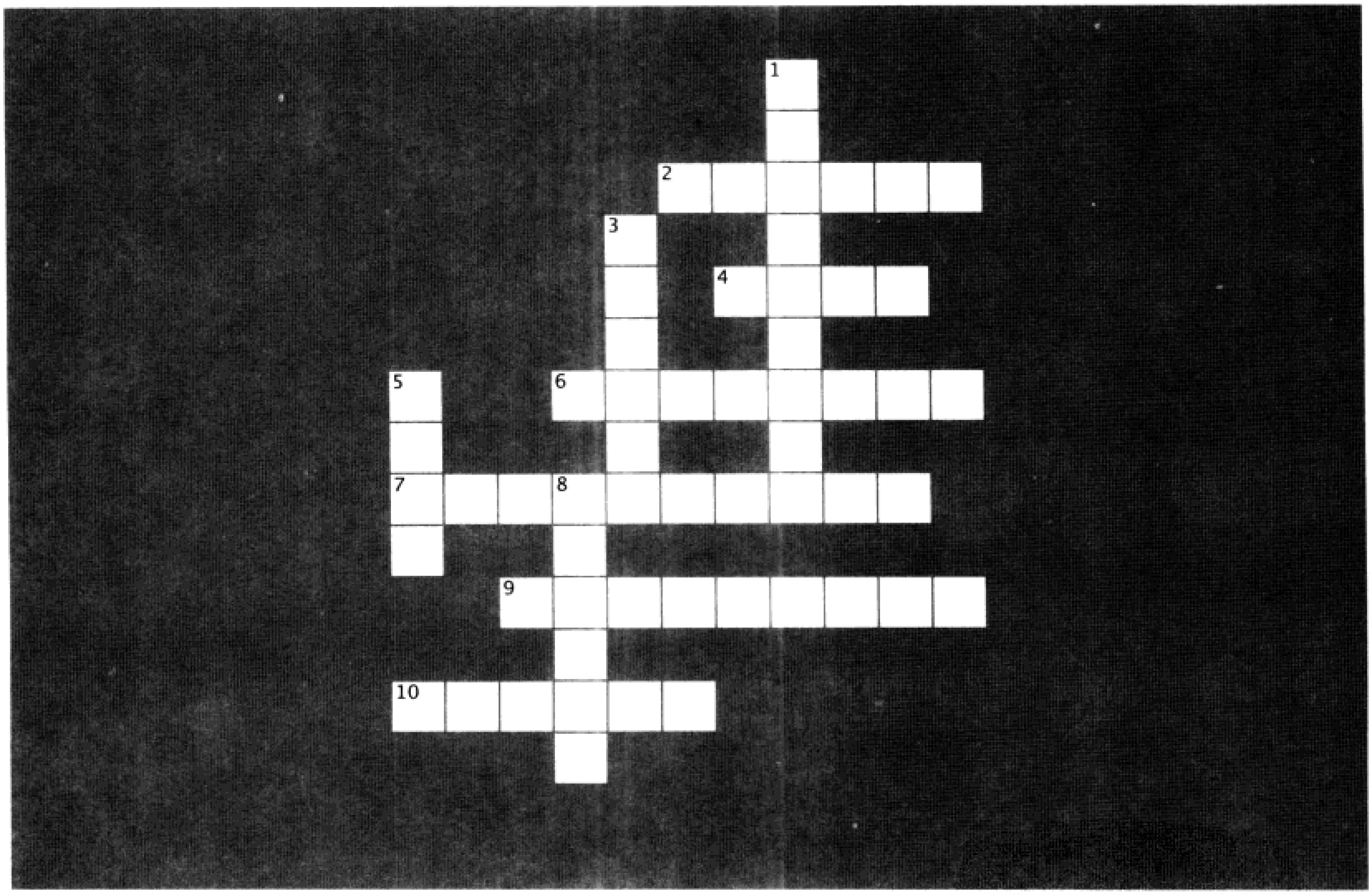


正义必胜!

【拼吾爱-最新编程资源的分享下载站】

添加功能纵横字谜

最后一次活动你的大脑右半球……



横向提示

2. UIImagePickerController可以从＿＿＿＿＿和照片库中获取图像。

4. ＿＿＿＿＿动画是UIKit自带的。

6. 圆形信息按钮只不过是一个配置好的＿＿＿＿＿。

7. 要使用地图开发包和Core Location，需要额外的＿＿＿＿＿。

9. 你的应用必须在＿＿＿＿＿上也能工作。

10. ＿＿＿＿＿表单是一种让用户在选项中进行挑选的好办法。

纵向提示

1. 照相机无法在＿＿＿＿＿中测试。

3. iPhone不是唯一可以运行应用的＿＿＿＿＿。

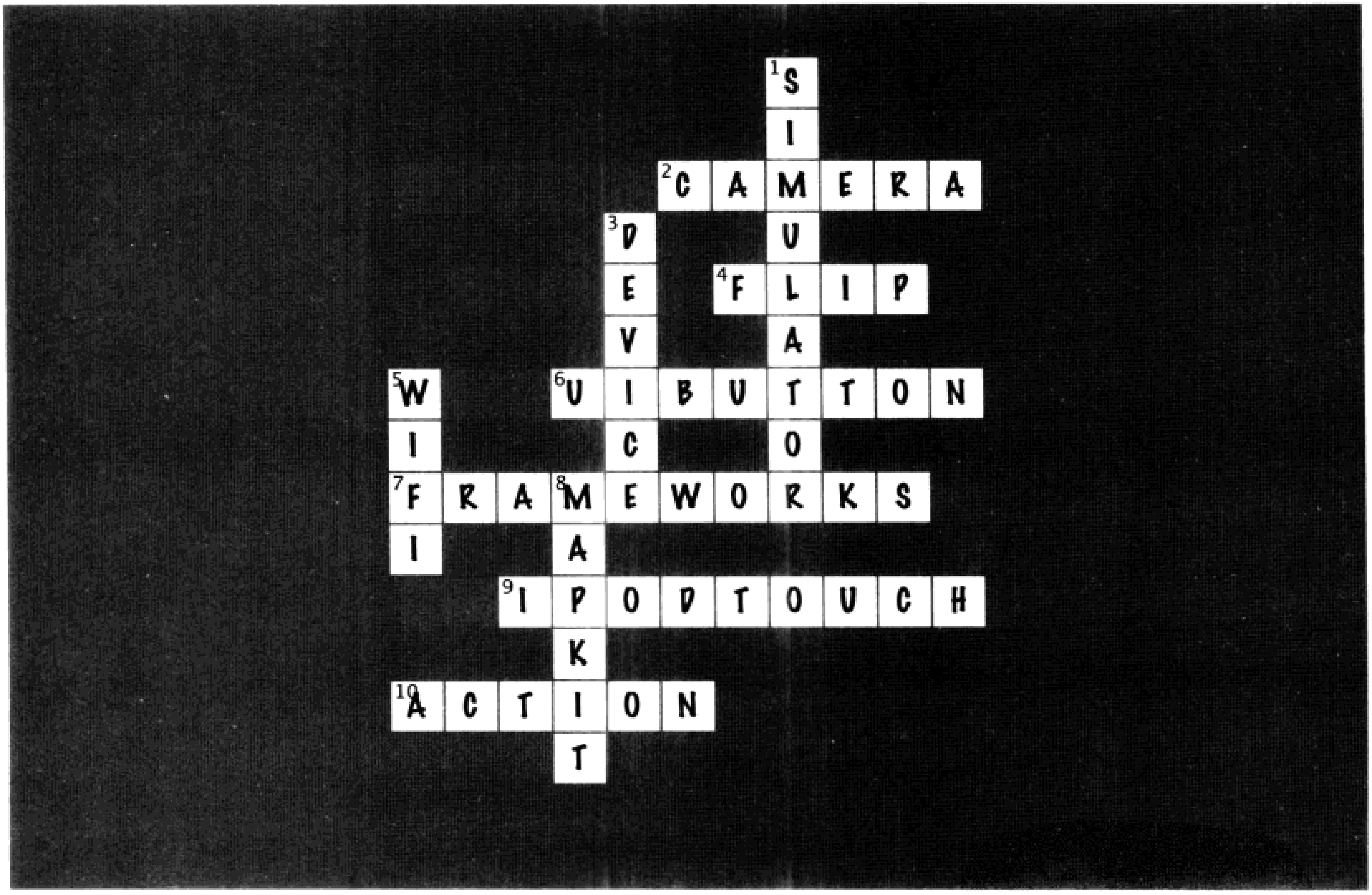
5. 除了GPS和基站定位，＿＿＿＿＿也可以用来确定位置。

8. 没有网络连接，＿＿＿＿＿无法正常工作。

【拼吾爱-最新编程资源的分享下载站】

添加功能纵横字谜答案

最后一次活动你的大脑右半球……



横向提示

2. UIImagePickerController可以从＿＿＿＿和照片库中获取图像。[CAMERA]

4. ＿＿＿＿动画是UIKit自带的。[FLIP]

6. 圆形信息按钮只不过是一个配置好的＿＿＿＿。[UIBUTTON]

7. 要使用地图开发包和Core Location，需要额外的＿＿＿＿。[FRAMEWORKS]

9. 你的应用必须在＿＿＿＿上也能工作。[IPODTOUCH]

10. ＿＿＿＿表单是一种让用户在选项中进行挑选的好办法。[ACTION]

纵向提示

1. 照相机无法在＿＿＿＿中测试。[SIMULATOR]

3. iPhone不是唯一可以运行应用的＿＿＿＿。[DEVICE]

5. 除了GPS和基站定位，＿＿＿＿也可以用来确定位置。[WIFI]

8. 没有网络连接，＿＿＿＿无法正常工作。[MAPKIT]

【拼吾爱-最新编程资源的分享下载站】



你的附加工具箱

你已经消化吸收了第9章，现在已经把照相机、Core Location和地图开发包加入到你的工具箱中了。如果你想得到一份完整的书中功能提示的列表，请访问http://www.headfirstlabs.com/iphonedev。

翻转动画

UIKit自带..

对于iPhone的工具应用来说是一种典型界面。

通常实现为模式视图。

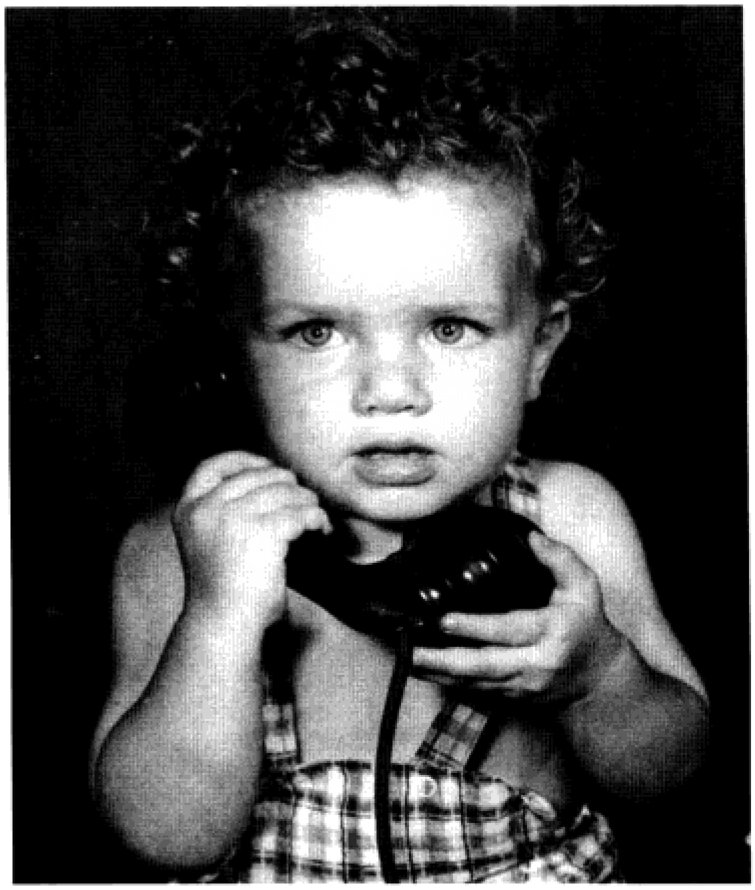
照相机

通过UIImagePickerController访问。

不是所有设备上都有的，你得处理这种情况。

允许你从库中直接选择并编辑一幅图像来用在应用中。

【拼吾爱-最新编程资源的分享下载站】



有你相伴真好！

我们看着你离去是如此难过， 但什么也比不上让你学以致用。你就要开始自己的iPhone征程，而我们已经将未来放进你的手中。离开前看一看附录吧，在那里，你会找到造就杰出iPhone应用之法，还有进入iTunes App Store的途径。我们渴望着听到来自你的消息，所以请光临Head First实验室的网站（http://www.headfirstlabs.com/iphonedev），让我们知道iPhone开发为你开拓了“钱程”！

【拼吾爱-最新编程资源的分享下载站】

i 子余

首要的六件事（我们未涉及的）



是否觉得还缺了某些东西？我们了解你的感受……就在你以为结束了的时候，才发现还有些事没有做。我们离开你之前，必须再说明几项额外的内容，这些内容我们没法放进本书的其他地方。至少，如果你还想随身带着本书，而不是用一个带轮子的金属箱来盛放，这些内容就没法进入正文。所以请看一看，有没有什么是你还不会的。

【拼吾爱-最新编程资源的分享下载站】

#1. 国际化与本地化

iPhone和iPod Touch在超过80个国家发售，自带对30种语言的支持。你应该根据你的应用程序，考虑支持多种语言和文化。国际化的过程，就是确认你的应用程序中和语言文化相关的部分，将其打造成支持多种语言环境的应用。你要注意的方面包括：

- Nib文件（视图、标签、按钮文字等）
- 与地点和文化相关的图标和图像，比如国旗和文字
- 内置或者在线的帮助信息和文档
- 应用程序中的静态文本



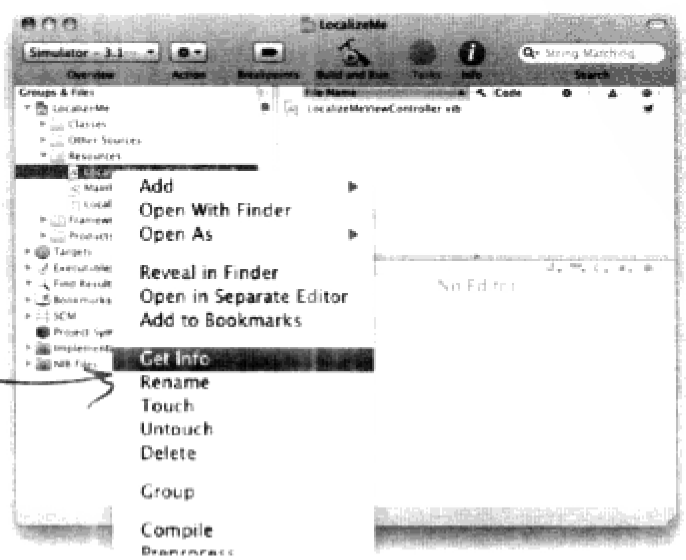
你可以在iPhone上改变语言和环境：进入Settings→General→International即可。

一旦你确定了自己的应用程序中与文化或者语言相关的部分，下一步就是对其进行本地化。iPhone OS对资源本地化的支持很强大，可以将本地化资源与应用程序的其他部分隔离开，让你可以轻松地利用本地化团队或者外包进行此项工作。

到目前为止，我们的资源都是包含在应用程序的.app目录中的。当你开始对资源进行本地化后，Xcode会为你添加的每个本地化（语言环境）创建一个lproj目录，将语境相关的资源挪到这里面。例如，如果你同时为nib提供了英语和法语翻译文件，那么应用程序中就会有一个en.lproj（或者English.lproj）以及一个fr.lproj目录。

对nib进行本地化

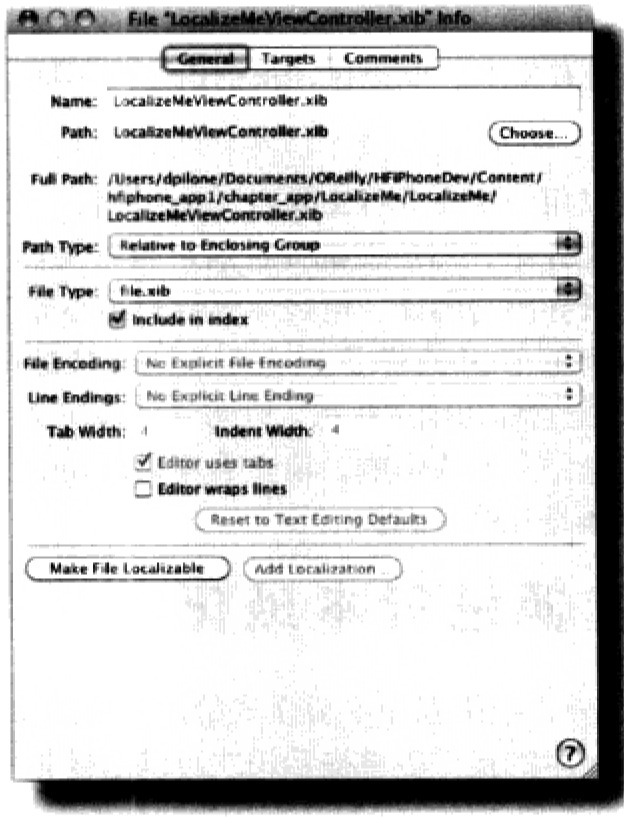
Xcode和Interface Builder对于nib本地化有内建的支持。在你开始着手翻译之前，需要让Xcode创建语言环境相关目录。



在你要本地化的nib上单击右键，然后选择“Get Info”。

【拼吾爱-最新编程资源的分享下载站】

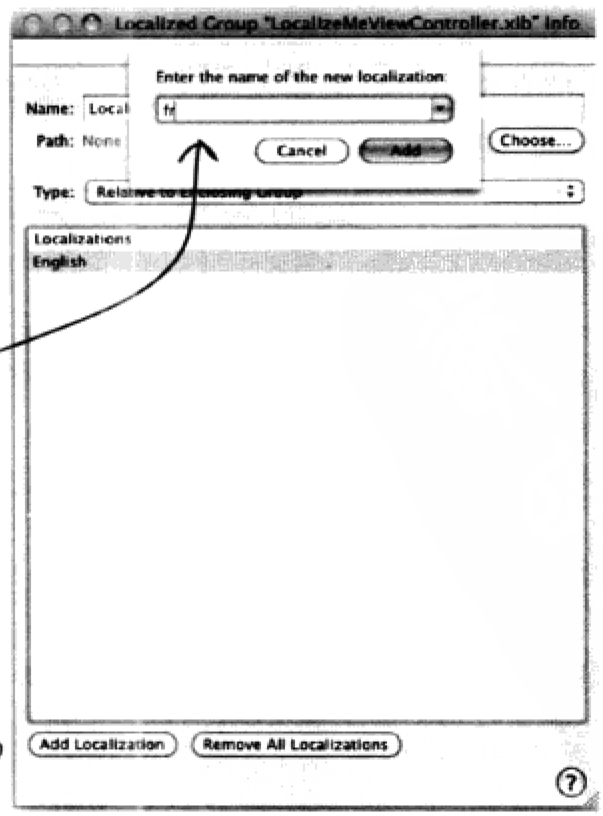
之后单击“Make File Localizable”，Xcode就会把你在项目列表中的nib条目转化为一个分组，每个本地化结果都列在下面。Xcode会把你原来的nib复制到默认的本地化结果中。



在“Make File Localizable”上单击，就会让Xcode创建基于环境的目录结构，从而来组织你的资源。

这就是nib的“Get Info”对话框，不过用这种方法可以本地化任何通用资源，比如图标或者图像。

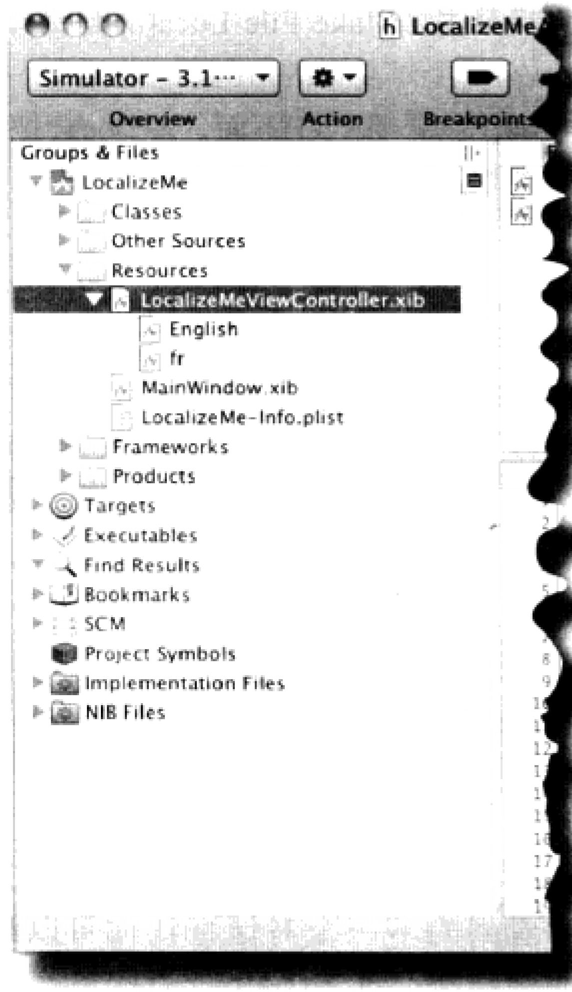
你看到的下一个对话框让你可以添加额外的本地化结果。选择**General**标签，并单击“Add Localization”。在出现的对话框中，你应当输入你要添加的本地化结果的国家代码。在我们的例子中，要添加的是法语。



“添加本地化”按钮会问你新建本地化的名字。请使用2～3个字母的国家代码，这些代码可以在苹果公司的文档中找到。不要用下拉列表里的值。

【拼吾爱-最新编程资源的分享下载站】

现在，对进行nib本地化需要做的工作，就只剩下在你要进行本地化的语言上双击，然后翻译所有文本。要记住，根据语言的不同，你可能还需要调整界面布局。



对于大型项目，可以用一个命令行工具ibtool，将一个nib中的所有字符串导出到一个外部文件中，然后在翻译过后，将结果再重新合并回nib中。这样就可以进行批量导出与翻译，但是你要特别小心界面布局的问题，因为这时候你没法看到每个nib的样子。一旦一个nib被翻译完毕，你就可以用Interface Builder把它标为锁定状态，以防意外地对文字或者布局进行修改，从而对你的翻译结果造成影响。请查看苹果公司的文档，里面有更多关于对束和nib进行本地化的信息。

对字符串资源进行本地化

除了nib中的文本之外，应用程序中要显式给用户看的文字也要做本地化。例如，iBountyHunter中的操作表单提示用户可以选择拍照、选取现有照片以及取消操作。这些按钮上的文字都是编程中生成的，需要被正确地翻译。

这种类型的文字被称为字符串资源，iPhone OS会为其使用一种字符串文件（strings file）。通常你支持的每一种语言，都会有一个这种文件。每个文件都包含有一段描述信息说明字符串要表达的意思、默认语言版本的字符串以及翻译之后的版本。如下所示：

```
/* Confirms a really bad decision. */
"All In" = "All In";

/* Cancels the dialog */
"Cancel" = "Cancel";

/* Title for the important alert view */
"This is important!" = "This is important!";

/* Warns the user about impending badness. */
"This will empty your bank account. Are you sure?" = "This will empty
your bank account. Are you sure?";
```

每个字符串资源都可以有一段描述，帮助翻译者理解字符串的上下文。

每个字符串都有原始串和翻译结果。

【拼吾爱-最新编程资源的分享下载站】

生成字符串文件

你可以手工创建字符串文件，但是让Xcode为你生成文件要容易得多。Xcode生成文件的方法是查找本地化宏，用它来加载翻译过的文本。要支持本地化字符串，你应该使用NSLocalizedString宏之中的一个，如下所示：

NSLocalizedString的第一个参数在翻译文件中用作关键字。通常这就是默认语言的字符串。

第二个参数就是和字符串一起显示在字符串文件中的注释。

```
- (IBAction) pushMePressed: (id) sender {
    UIAlertView *alertView = [[UIAlertView alloc]
      initWithTitle:NSLocalizedString(@”This is important!”,
                                      @”Title for the important alert view”)
      message:NSLocalizedString(@”This will empty your bank account.  Are you sure?”,
                                @”Warns the user about impending badness.”)
      delegate:nil
      cancelButtonTitle:NSLocalizedString(@”Cancel”, @”Cancels the dialog”)
      otherButtonTitles:NSLocalizedString(@”All In”,
                                          @”Confirms a really bad decision.”),
      nil];
    [alertView show];
}
```

如果你在代码中使用了NSLocalizedString宏，就可以很简单地生成字符串文件了，只要在命令行上运行genstrings命令就行了，如下所示：

```
genstrings -o English.lproj *.m */*.m
```

你需要为你支持的每一种翻译都运行一遍这条命令。这样就会在指定的语言目录下创建出一个名为Localized.strings的文件，你可以把这个文件交给译员。你还得把这个字符串文件添加到Xcode项目中，就像任何其他资源一样，不过只要它存在，iPhone OS就会在运行时根据用户为设备选择的语言，在相应的文件中寻找字符串。

iPhone OS提供的本地化功能十分强大，包括对货币、时间以及时期表示方式的支持；我们还只是了解了一点表面。苹果公司提供了几个关于国际化与本地化的文档，其中就有Xcode文档中的《Introduction to Internationalization Programming Topics》，可以帮助你处理更加复杂的情况。



留神!

iPhone OS会缓存资源！

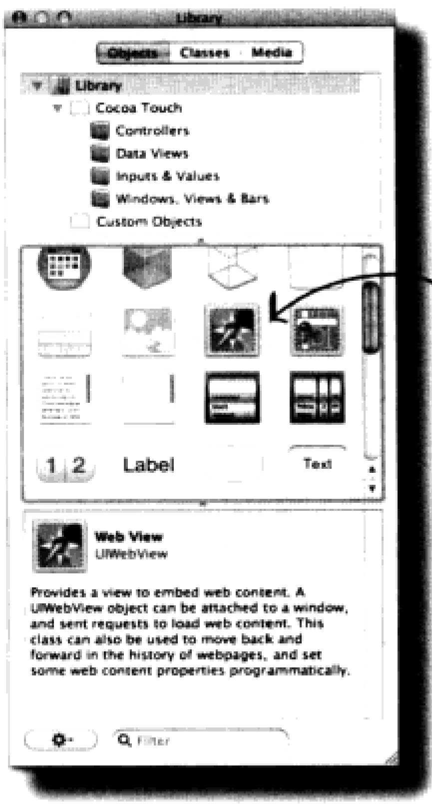
如果你已经安装过没有进行翻译的应用，iPhone OS很有可能会将资源缓存起来，这样即使添加了翻译过的资源，你也不会看到它们，除非你先卸载应用然后再安装！

【拼吾爱-最新编程资源的分享下载站】

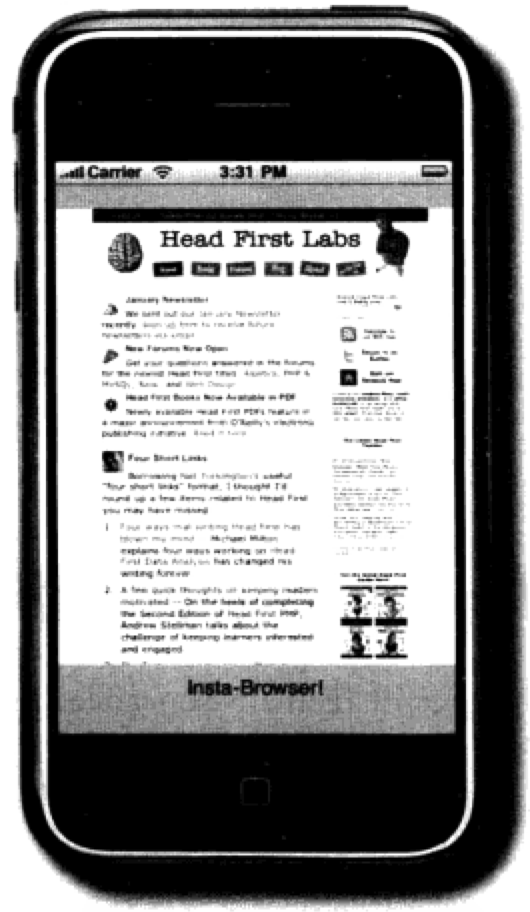
#2. UIWebView

iPhone OS自带了一个功能强大的控件，名叫UIWebView，它可以用Web Kit引擎来处理网页内容。基本上它就是一个封装的Safari浏览器。你可以用这个控件像正常浏览器一样加载外部URL，也可以加载本地内容来显示HTML格式的文档。虽然它的功能如此强大，但它却仍然是最容易使用的控件之一。

要创建一个UIWebView很简单，只要Interface Builder中将它拖放在视图中就可以了，然后在视图控制器中为它设置一个输出口。



只要把UIWebView控件拖放到你的视图上，就可以使用这个强大的浏览器组件。



使用UIWebView

UIWebView非常易于使用。要加载一个URL，只要向它发送一个loadRequest:消息，带上你要让它加载的URL就可以了，如下所示：

```
NSURL *url = [NSURL URLWithString:@"http://www.headfirstlabs.com"];
NSURLRequest *request = [NSURLRequest requestWithURL:url];
webView.scalesPageToFit = YES;
[webView loadRequest:request];
```

用我们需要的URL来初始化一个NSURL。

我们要让整个页面一开始就全部显示出来，所以我们打开了scalesPageToFit。

【拼吾爱-最新编程资源的分享下载站】

子余

UIWebView的属性

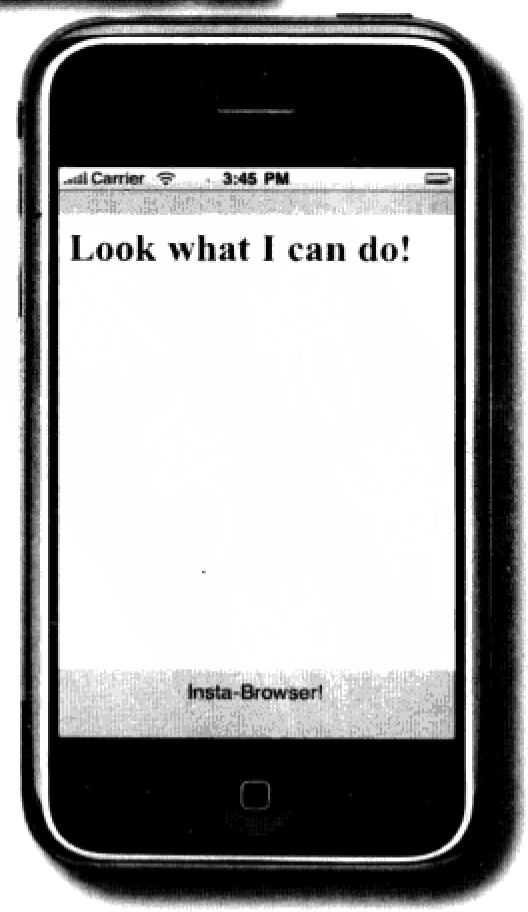
加载一个URL之后，你就可以用loading属性发现UIWebView当前正在加载URL。要让它停下来，可以向其发送stopLoading消息。要控制页面的显示方式，有几种选择。你可以关闭detectsPhoneNumbers属性，让它忽略正在显示的页面中的电话号码（否则它就会把这些号码转变成超链接，链到电话应用程序）。默认情况下，UIWebView会显示完整尺寸的页面。然而，你可以打开scalesPageToFit属性，让它将URL的内容进行缩放以适应屏幕。如果这个属性被开启，用户可以用通常的捏合手势来对内容进行放大和移动。

UIWebView也内建了对导航历史记录功能的支持。它会根据在向前、向后的历史中是否有页面存在，而设置自己的canGoBack和canGoForward属性。如果你想要支持导航，通常会用这些属性来打开或关闭前进、后退按钮。UIWebView知道历史记录是什么样的，所以你可以简单地向它发送goBack:或goFoward:消息即可，剩下的工作它会自己完成。

加载生成出来的内容

你也可以用UIWebView来加载本地生成的内容（比如显示HTML的帮助文件或者报告），只要请求它加载一个HTML字符串就行了，如下所示：

```
NSString *html = @"<html><body><h1>Look what I can do!</h1></body></html>";
[webView loadHTMLString:html baseURL:[NSURL URLWithString:@"file:///."]];
```



UIWebView同样支持代理

如果你想进一步了解UIWebView中发生的事，可以设法遵循UIWebViewDelegate协议，并为你的网页视图设置代理。代理协议让你能在加载开始和结束时得到通知，同样也会有机会在进行链接跳转之前检视链接。如果UIWebView有代理，用户点击链接时，它会在实际跳转到链接之前，向代理发送webView:shouldStartLoadWithRequest:navigationType消息。你可以返回NO，这样网页视图就不会跳转到该URL。

【拼吾爱-最新编程资源的分享下载站】

#3. 设备方向和视图旋转

在表面上，iPhone OS让处理屏幕旋转变得简单。iPhone和iPod Touch都含有一个加速计，让设备可以探测方向。当你用UIKit构建应用程序时，iPhone OS会询问当前活动的视图控制器是否可以处理旋转。iPhone OS支持下列方向：

界面方向常量	描述
UIInterfaceOrientationPortrait	通常的方向，主按钮在下方。默认情况下，这是视图控制器唯一支持的方向。
UIInterfaceOrientationPortraitUpsideDown	与纵向类似，但是主按钮是在设备的上方。
UIInterfaceOrientationLandscapeLeft	设备被侧向持握，主按钮在右侧。
UIInterfaceOrientationLandscapeRight	设备被侧向持握，主按钮在左侧。

视图控制器告诉iPhone OS自己支持哪些方向

当iPhone OS探测到设备被旋转到上面这些方向之一的时候，会调用当前活动视图控制器的`shouldAutorotateToInterfaceOrientation:`方法，并将新的方向作为参数传入。如果你的视图能够在给定方向下使用，可以简单地返回YES。如果不能使用，则返回NO。如果你没有显式地实现这个方法，默认实现会对除`UIInterfaceOrientationPortrait`以外的所有旋转方向返回NO，。

当iPhone OS需要旋转到一个新的方向时，会通知视图控制器，发送给它一个willRotateToInterfaceOrientation:消息，带有转场动画的时长作为参数。你可以用这个方法关闭一些在视图变换后会引起问题的东西，比如按钮或者计时器等。当动画结束之后，你就会收到didRotateFromInterfaceOrientation:消息，在这里可以重新开启刚才关闭的所有东西。

iPhone模拟器也支持旋转，所以你可以用每个方向来测试你的应用程序。要旋转模拟器，可以用Hardware→Rotate Right（或者Left）菜单项，或者按键⌘→（或者左箭头）。

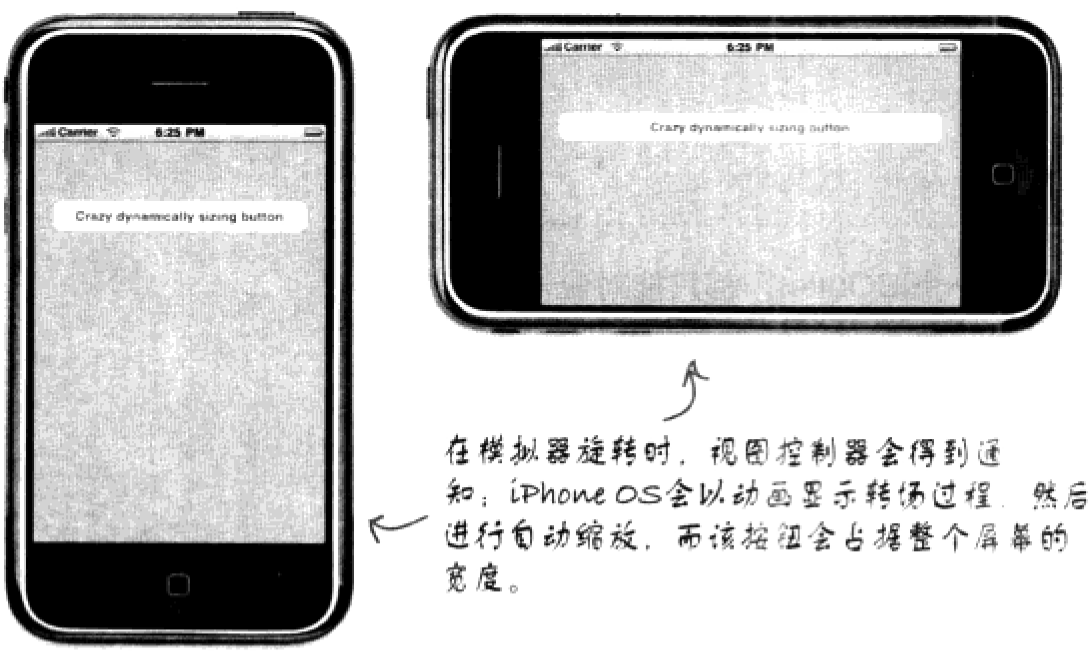
【拼吾爱-最新编程资源的分享下载站】

子余

处理视图旋转

处理视图旋转的最简单的方法，就是利用UIKit的能力自动改变控件的大小。要做到这一点，需要选中控件然后启动查看器，再来到标尺页（按⌘3）。在这里，你可以选择自动缩放的锚点，就是控件有哪些边会原地不动。通过配置自动缩放信息，你可以让UIKit在视图大小改变时，随之自动改变控件的大小。





但是有些时候自动缩放不够用……

【拼吾爱-最新编程资源的分享下载站】

用两个不同的视图来处理旋转

随应用程序的不同，视图有可能会很复杂，以致自动缩放无法达到你在旋转视图时想要的效果。反其道而行之，有些应用程序会在横向模式下向用户呈现一个与纵向模式下完全不同的外观。



内建的股票应用程序会根据设备方向而显示两个不同的视图——这可不是对控件进行自动缩放就能做到的。



要支持多个视图，你需要在nib中定义多个UIViews或者创建单独的nib。然后，在视图控制器得到旋转的通知时，就可以改变你的`self.view`，将其设置为与要使用的方向相对应的视图。



对于任何一个IBOutlet，都只能对应一个对象！

如果你的视图中有类似的控件（但是位置不同或者风格有异），你得为每个控件准备单独的输出口。

【拼吾爱-最新编程资源的分享下载站】

#4. 视图动画

如果你已经用过一段时间iPhone或者iPod Touch，就会知道平滑的转场和优雅的动画是苹果的用户体验的招牌效果。目前在我们打造的应用程序中，还仅仅涉及几种基本的动画（比如iBountyHunter中用到的翻转动画）。其实，从添加删除表格行到在屏幕上滑动控件，几乎每件事都可以变成动画。

动画表示表格视图更新

如果你打算在一个表格视图中添加或删除多行，可以要求它使用一种平滑的动画效果进行（也是一种更高效的处理表格视图更新的方式），只要在你开始操纵数据前向它发送一个beginUpdates消息就行了，然后在结束之后再发送一个endUpdates，如下所示：

```
[self.tableView beginUpdates];
[self.tableView insertRowsAtIndexPaths:insertIndexPaths
      withRowAnimation:UITableViewRowAnimationRight];
[self.tableView deleteRowsAtIndexPaths:deleteIndexPaths
      withRowAnimation:UITableViewRowAnimationFade];
[self.tableView endUpdates];
```

插入多行时，你可以用insertRowsAtIndexPaths告诉表格视图要添加的新的索引路径。表格视图会立即向数据源和代理请求这些新行的单元格信息，如果你指定了动画类型，它们会平滑地滑动进入表格。

beginUpdates和endUpdates会告诉表格视图你要进行多行修改，这样它就不会实际显示任何动画，直到进行endUpdates调用为止；然后所有动画（插入或删除）就会立即开始。

动画表示视图和控件的变化

与表格视图类似，UIView也有内建的支持，可以将它的几种属性的变化以平滑的动画来显示。你只要告诉视图想要让它以动画表示一项改变就可以了，方法是向视图发送beginAnimations消息，描述变化的终点，然后通过发送commitAnimations消息请求开始转场过程。下面的UIView属性都可以自动以动画方式表示：

UIView属性	描述
frame	实际对视图进行描述的矩形——视图的起点和尺寸——在父视图的坐标系中。
bounds	在本地坐标系中，视图的起点和尺寸。
centerpoint	视图在父视图坐标系中的中心位置。
transform	应用到视图上的任何形变（旋转、变换等）。
alpha	视图的透明度。

【拼吾爱-最新编程资源的分享下载站】

#5. 加速计

在iPhone和iPod Touch中，加速计是最多能的硬件设备之一。加速计令设备可以监测到加速以及沿3个轴向的重力。只要短短几行代码，你就可以辨别设备是右侧朝上、上下颠倒还是平放于桌面上，等等。你甚至可以监测到设备改变方向的速度。

你所需要的就是UIAccelerometer

从设备获取方向信息很直接了当。有一个共享的UIAccelerometer实例可以让大家访问。就像许多其他iPhone OS类一样，UAccelerometer也有一个代理协议UIAccelerometerDelegate，其中声明了一个方法可以接收加速度信息。你用来接收加速信息的类应该遵循这个UIAccelerometerDelegate协议，并实现didAccelerate:方法：

```
- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration;
```

你会得到一个对加速计的引用，以及一个UIAcceleration类的实例，其中含有实际的加速度信息。

要接收加速度信息，你只需要告诉加速计代理是谁，发送加速度信息的频率又是多少，如下所示：

```
self.accelerometer = [UIAccelerometer sharedAccelerometer];
self.accelerometer.delegate = self;
self.accelerometer.updateInterval = 0.5f;
```

获得共享的加速计实例…

…然后配置代理和以秒为单位的更新频率。我们要求每秒2次更新。

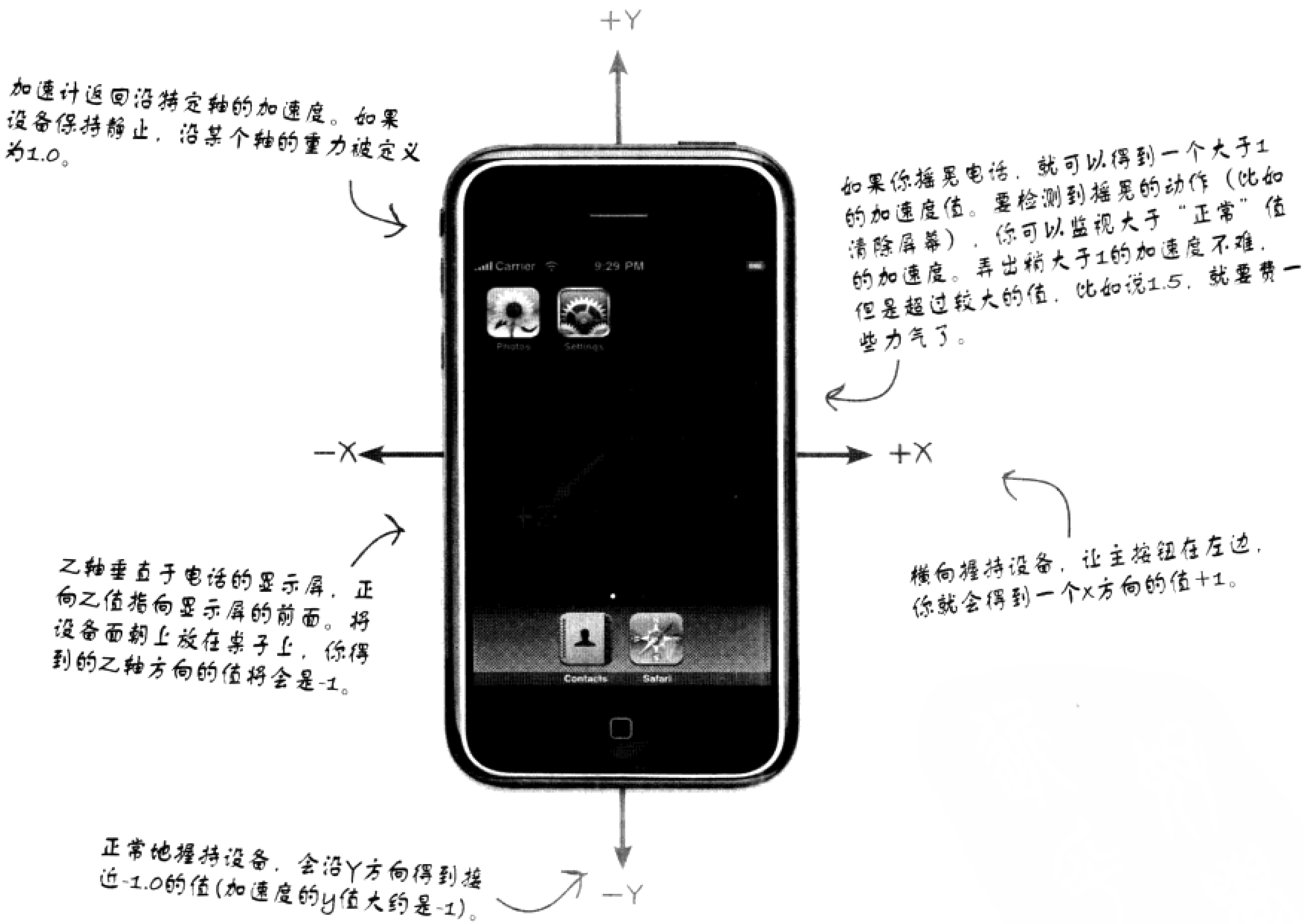
每个UIAcceleration对象都包含有沿x/y/z 3个轴方向的加速度信息和一个数据采集的时间戳。给一个简单的例子，你可以根据加速度信息更新标签，如下所示：

```
- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration {
        self.xOutput.text = [NSString stringWithFormat:@"%.4f", acceleration.x];
        self.yOutput.text = [NSString stringWithFormat:@"%.4f", acceleration.y];
        self.zOutput.text = [NSString stringWithFormat:@"%.4f", acceleration.z];
}
```

【拼吾爱-最新编程资源的分享下载站】
子余

理解设备加速

首先是一个坏消息。模拟器根本不能模拟加速计。不管你多厉害地摇晃Mac，也无法得到任何信息。你得将应用程序安装在真实设备上，才能获得实际的加速计信息。不过只要你这样做了……



如果你要构建的是一个常规的基于视图的应用程序，UIKit会通知你方向的改变，并在用户晃动电话时自动提供撤消/重做功能，从而隐藏了大量对加速计的需求。加速计对于自定义绘制的应用程序最为有用，比如游戏（操纵方向或平衡）以及工具应用程序（水平仪）。

【拼吾爱-最新编程资源的分享下载站】

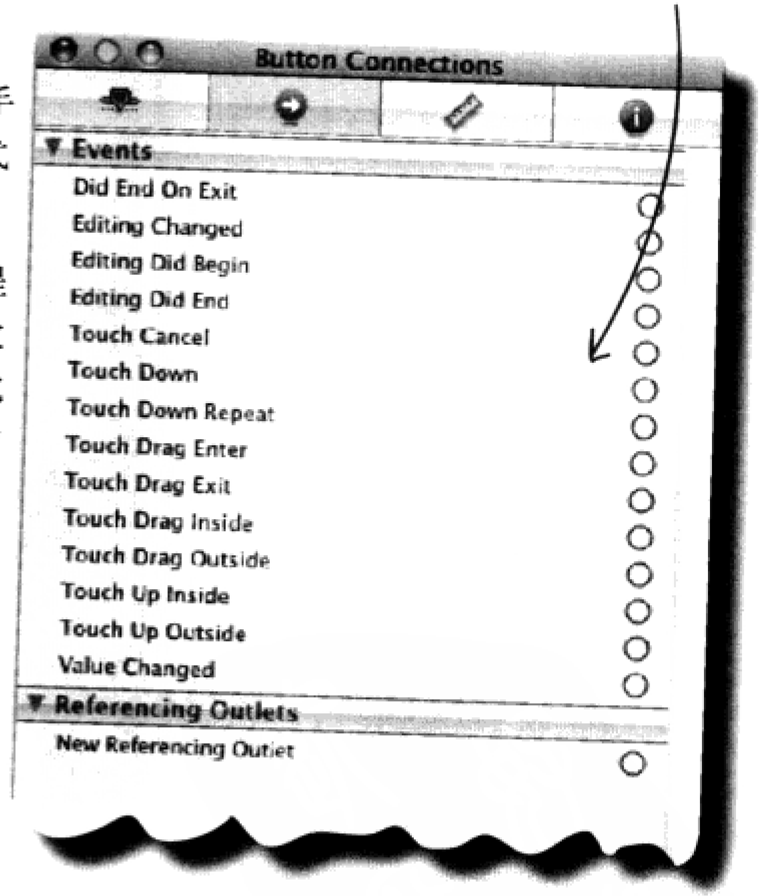
#6. 关于游戏的只言片语……

iPhone上的游戏有巨大的市场，许多人也都玩过，但是游戏同样也是相当高级的应用程序。详细讲解这样的应用程序超出了本书的范畴——这些应用会使用多点触摸交互方式、Quartz和OpenGL图形库，以及P2P网络——但是在这里我们会向你简单地介绍一下可能会用到的技术，以及哪里能找到更多这方面的信息

多点触摸

你很可能已经注意到了，我们只提到了一种可能在应用中被触发的按钮事件**touch up inside**。iPhone有能力同时检测到多达5个手指的触摸，并可将这些手指与屏幕的互动方式解释成多种不同类型的事件。

这些就是可以触发的全部按钮事件。



除了触摸之外，iPhone还可以检测到扫动和其他能够配置的手势。通过定义扫动的长度和方向，你可以创建许多不同的方式来与你的应用程序进行互动。

捏合就是一个自定义手势，苹果公司在许多自己的默认应用程序中都用到了它，最引人注目的就是Safari中用来对视图进行放大缩小。要做的只是注册接收两个手指的触摸事件，并跟踪它们之间的距离变化：如果距离增加，就放大，如果减少，则缩小。

使用这些事件，意味着你可以为用户创建自定义界面，而不只是触摸按钮。要使用多点触摸，需要将你的视图配置成一个多点触摸视图，然后还需要编码处理你感兴趣的各种不同类型的事件。

使用这些事件要求你利用响应者链（见`UIResponder`类和`UIEvents`类的参考资料）。

【拼吾爱-最新编程资源的分享下载站】

Quartz和OpenGL

Quartz和OpenGL，是两种在iPhone上创建图形的方式，二者都很庞大，足够自己写一本书了，不过这里会简要地对它们做一下介绍。

Ed提示：这可是一个好主意…

Quartz

在二者当中，Quartz比较简单一些，可以让你直接在视图中进行二维绘图。绘图代码会使用`Core Graphics`框架，并直接在视图中进行渲染。它遵循的是一种画家模型，也就是说命令的次序是很重要的。首先绘制的东西会被随后在同一位置上的绘图动作所覆盖。Quartz可以处理阴影、色彩，并可以操作其他类型的图像和视频。

开发者文档中的《Quartz 2D Programming Guide》中有许多信息，可以帮助你从初学者入门。

OpenGL

OpenGL可以用于二维或三维图形，而且要复杂得多得多，但是这也意味着你使用起来灵活度更大。它是一种完善的跨平台库，在移动设备上的实现称为OpenGL ES，需要通过OpenGL ES框架使用。

你可以用它来绘制线条、多边形和物体，也可以让它们变成动画。开发者文档中有一篇很不错的入门教材，就是《OpenGL ES Programming Guide for iPhone OS》。

游戏开发包（GameKit）

在iPhone OS 3中新出现的GameKit框架，让你可以使用P2P网络以及蓝牙语音来帮助游戏，在玩的过程中与其他设备进行互动。这些功能在第一代iPhone、iPod Touch和模拟器上不存在。

有一个与图像选取器类似的`GKPeerPickerController`，可以提供一个标准界面，来寻找其他同样运行你的应用的设备并建立连接。连接建立之后，你就可以在设备之间传输数据或语音了。

要想在你的应用中利用这一新功能，有一个入门的好地方就是《GameKit Programming Guide》。

【拼吾爱-最新编程资源的分享下载站】

【拼吾爱-最新编程资源的分享下载站】

ii 发布应用所需的预备工作

准备好进入App Store



你想让你的应用进入App Store，对吗？目前我们基本上还是在模拟器中使用我们的应用，这没问题。但是要想进入下一阶段，你还需要将一个应用安装在实际的iPhone或者iPod Touch上，然后才能申请将它放进App Store中。而要想做到这一点，唯一的办法就是成为苹果公司的注册开发者。即便如此，要把你写的应用放到自己的个人设备上，也不是在Xcode里按个按钮那么简单的事。为了这个目标，我们该和苹果公司谈谈了。

【拼吾爱-最新编程资源的分享下载站】

苹果公司指定了规则

我们曾经讲到过HIG，以及苹果公司的审查过程会多么严格——他们要保护自己的平台。这项工作的一部分，就是监视你的iPhone上运行的程序，即使这东西是你自己写的也不例外。

这里我们会向你大致描绘一下如何将一个应用放到你的设备上，然后接下来就是准备提交。我们不可能对整个过程面面俱到——总之你得成为iPhone开发者计划的会员，并交纳99美元的费用。

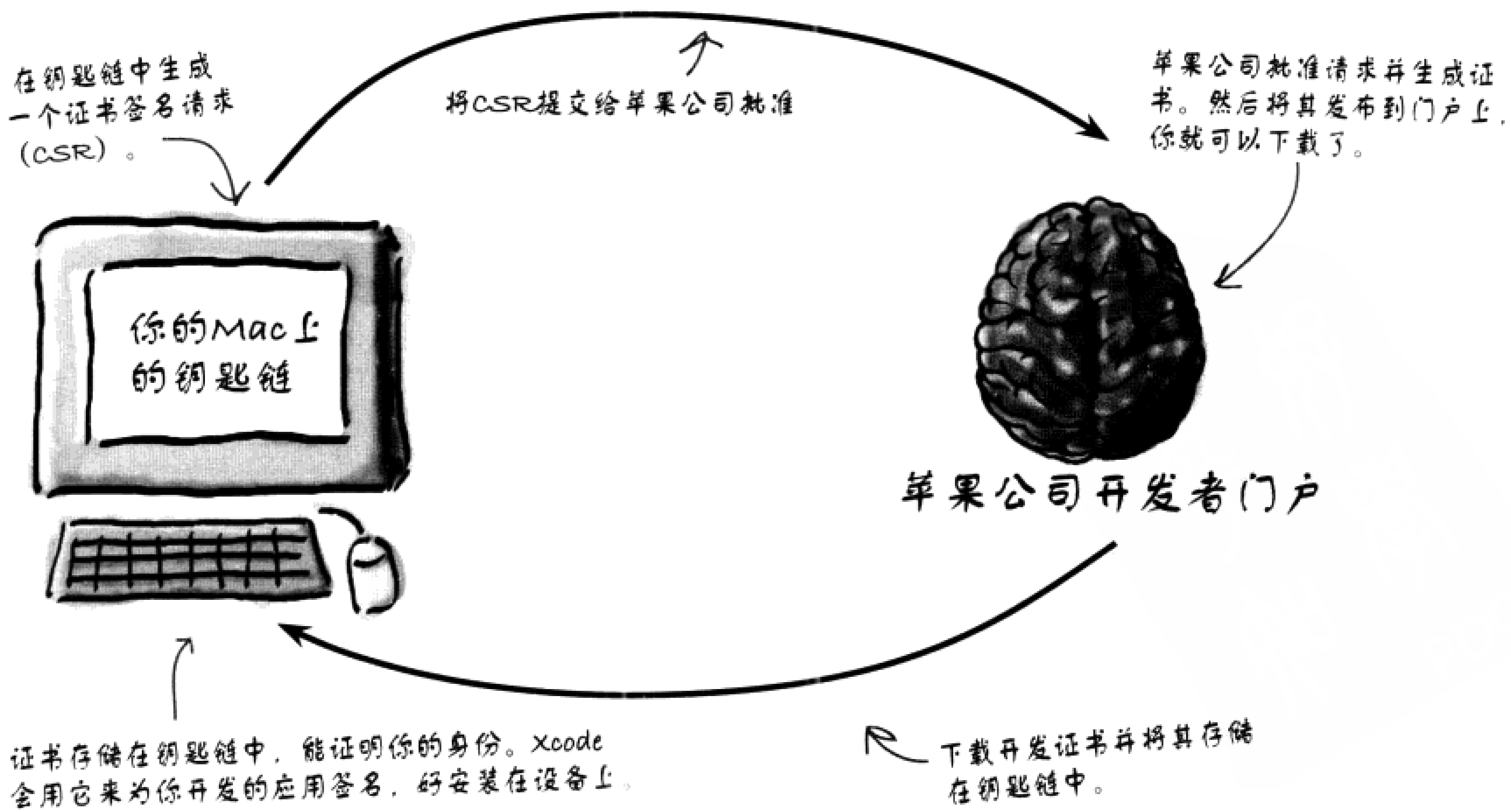
Xcode文档中的《iPhone开发指南》中有一些更好的信息，你可以在加入开发者计划前看一看。

从苹果公司的开发者门户出发

开发者门户，也就是你第一次下载SDK的地方，同时也是一个管理中心，可以管理你所有的电子签名，你需要使用这些签名，才能将应用安装并运行在你的iPhone上。

首先获得你的开发证书

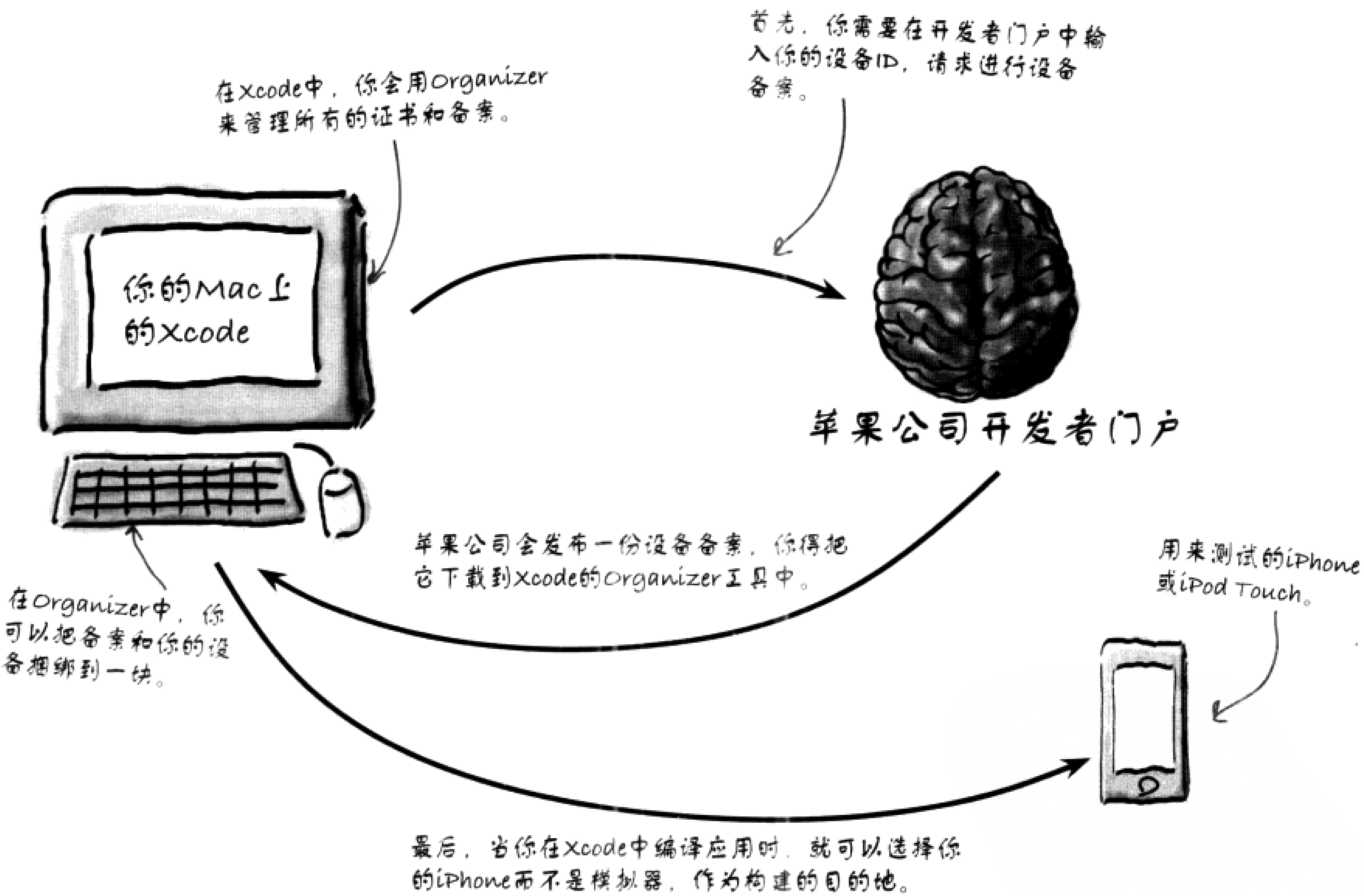
要想把你的应用从Xcode中安装到一部iPhone或者iPod Touch上进行测试，整个过程需要用到一个开发证书和设备备案。这个证书是由你和苹果公司共同签署的，证明你注册成为一个开发者。它创建了一个公钥和一个私钥，而这个私钥存储在你的Mac上的钥匙链中。下面就是让证书工作起来的过程。



【拼吾爱-最新编程资源的分享下载站】

设备备案将所有这些整合到一起

现在你的开发证书已经到位了，要完成整个流程你还需要一份设备备案。这份电子文档可以将应用（通过iPhone应用程序ID）、开发者以及证书捆绑在一起，以便在设备上进行安装。





如果你没有开发证书，是无法获得设备备案的。

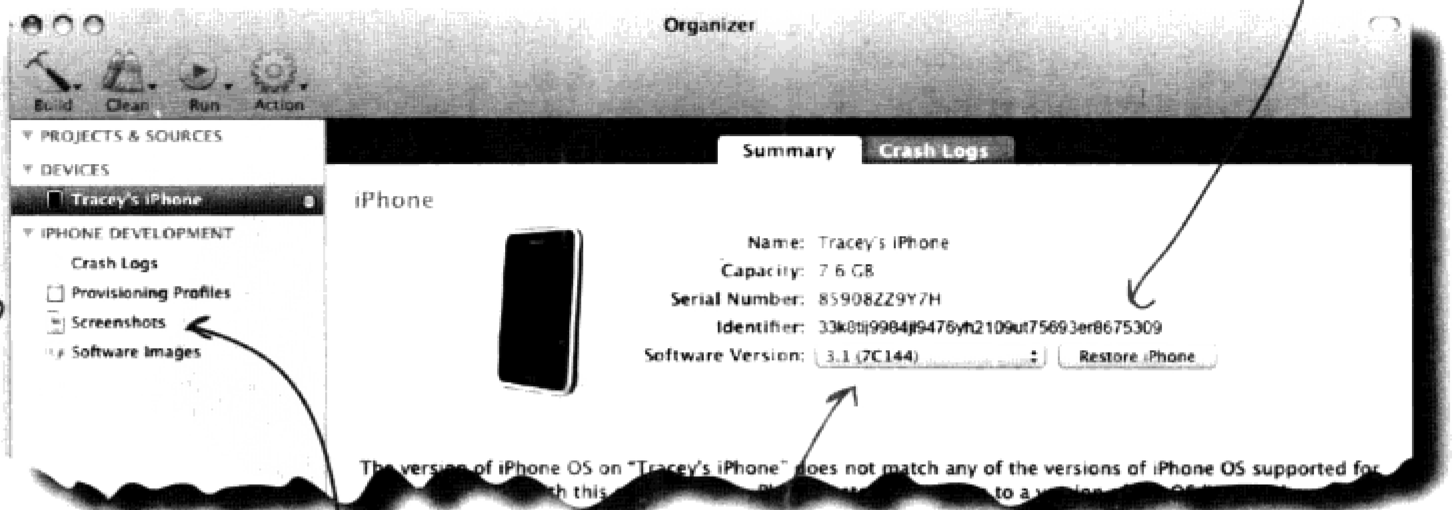
【拼吾爱-最新编程资源的分享下载站】

在Organizer中监视状态

Organizer是一个Xcode自带的工具，以前我们没怎么讨论过它，但它却是让这一整套电子文牍保持井井有条的关键所在。在Xcode中，从菜单中选择**Window→Organizer**选项，就可以打开它。

如果你的iPhone插到电脑上，就会看到类似的场面。

设备备案要的就是这个标识符，这对每个设备都是唯一的。



在这里你可以看到你的设备备案。

你还可以通过Organizer做屏幕抓图。

在这儿可以配置你的设备用于开发。Organizer会确保你的软件版本对于开发工作是有效的，如果无效，它会帮你选一个可以用的。

最后几点提示……

这个快速概述应该让你对这个流程是如何工作的有了一些概念，但是你还需要加入开发者计划才能了解全部细节。我们的目的只是帮助你对这个过程有个整体印象。

有几件事你得明白。首先，当你作为团队成员进行开发时，团队的管理者必须参与这些步骤。其次你需要完成整个流程，才能在你的设备上安装应用，不管你是不是打算把它向全世界发布。

最后一点，怎么进入App Store？一旦你加入了开发者计划，而应用程序也进行了测试，那么你就可以把它提交进行审查了。

更多信息

加入开发者计划之后，你就可以进入开发者门户，找到**iPhone Development Program User Guide**。里面有许多不错的信息可以帮助你走完整个流程。